

GaussDB

Developer Guide(Distributed_V2.0-3.x)

Issue 01
Date 2025-03-10



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2025. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: <https://www.huaweicloud.com/intl/en-us/>

Contents

1 Database System Overview.....	1
1.1 Database Logical Architecture.....	1
1.2 Query Request Handling Process.....	2
1.3 Managing Transactions.....	2
1.4 Concepts.....	6
2 Database Security.....	8
2.1 Users and Permissions.....	8
2.1.1 Default Permission Mechanism.....	8
2.1.2 Administrators.....	9
2.1.3 Separation of Duties.....	11
2.1.4 Users.....	13
2.1.5 Roles.....	14
2.1.6 Schemas.....	16
2.1.7 User Permissions.....	17
2.1.8 Row-Level Security.....	18
3 Database Quick Start.....	21
3.1 Connecting to a Database.....	21
3.1.1 Using gsql to Connect to a Database.....	21
3.1.2 APIs.....	21
3.2 Operating a Database.....	22
3.2.1 Creating a Database Account.....	22
3.2.2 Creating and Managing Databases.....	23
3.2.3 Creating and Managing Tablespaces.....	25
3.2.4 Creating and Managing Tables.....	27
3.2.4.1 Creating a Table.....	28
3.2.4.2 Inserting Data to a Table.....	28
3.2.4.3 Updating Data in a Table.....	32
3.2.4.4 Viewing Data.....	33
3.2.4.5 Deleting Data from a Table.....	33
3.2.5 Querying System Catalogs.....	34
3.2.6 Other Operations.....	36
3.2.6.1 Creating and Managing Schemas.....	36

3.2.6.2 Creating and Managing Partitioned Tables.....	39
3.2.6.3 Creating and Managing Indexes.....	43
3.2.6.4 Creating and Managing Views.....	47
3.2.6.5 Creating and Managing Sequences.....	48
3.2.6.6 Creating and Managing Scheduled Jobs.....	50
4 Development and Design Proposal.....	54
4.1 Overview.....	54
4.2 Database Design Specifications.....	55
4.2.1 General Specifications.....	55
4.2.2 Deployment Specifications.....	57
4.2.3 Database Object Naming Specifications.....	58
4.2.4 Database and Schema Design Specifications.....	59
4.2.5 Permission Design Specifications.....	61
4.2.6 Character Set Design Specifications.....	62
4.2.7 Table Design Specifications.....	62
4.2.8 Column Design Specifications.....	68
4.2.9 Index Design Specifications.....	71
4.2.10 Function/Stored Procedure Design Specifications.....	73
4.2.11 Constraint Design.....	73
4.2.12 View and Joined Table Design.....	74
4.3 Database Programming Specifications.....	74
4.3.1 GUC Parameter Programming Specifications.....	74
4.3.2 Object Access Programming Specifications.....	74
4.3.3 WHERE.....	75
4.3.4 SELECT.....	77
4.3.5 INSERT.....	78
4.3.6 UPDATE.....	78
4.3.7 DELETE.....	79
4.3.8 Join Query.....	79
4.3.9 Subquery.....	80
4.3.10 Transaction.....	80
4.3.11 SQL Compilation.....	81
4.3.12 DDL.....	85
4.4 Client Programming Specifications.....	85
4.4.1 JDBC.....	85
4.4.2 Connection Control Specifications.....	88
5 Application Development Guide.....	89
5.1 GaussDB Application Development Guide.....	89
5.2 Development Specifications.....	93
5.3 Obtaining the Driver Package.....	94
5.4 Development Based on JDBC.....	95
5.4.1 Development Process.....	95

5.4.2 Development Process.....	96
5.4.2.1 Obtaining the JAR Package of the Driver and Configuring the JDK Environment.....	96
5.4.2.2 Connecting to a Database.....	99
5.4.2.2.1 Connection Methods.....	99
5.4.2.2.2 Connection Parameter Reference.....	101
5.4.2.2.3 Connecting to a Database in Non-Encrypted Mode.....	127
5.4.2.2.4 Connecting to a Database in SSL Mode.....	131
5.4.2.2.5 Connecting to a Database in UDS Mode.....	134
5.4.2.3 Running SQL Statements.....	134
5.4.2.4 Processing Data in a Result Set.....	139
5.4.2.5 Closing a Connection.....	142
5.4.3 Typical Application Development Examples.....	142
5.4.3.1 Configuring Connection Parameters in Different Scenarios.....	142
5.4.3.2 Creating and Calling a Stored Procedure.....	144
5.4.3.3 Obtaining the Return Value of a Function.....	146
5.4.3.4 Batch Query.....	147
5.4.3.5 SQL Retry at an Application Layer.....	149
5.4.3.6 Logical Replication.....	153
5.4.4 JDBC Interface Reference.....	160
5.4.4.1 java.sql.Connection.....	160
5.4.4.2 java.sql.CallableStatement.....	170
5.4.4.3 java.sql.DatabaseMetaData.....	172
5.4.4.4 java.sql.Driver.....	180
5.4.4.5 java.sql.PreparedStatement.....	181
5.4.4.6 java.sql.ResultSet.....	184
5.4.4.7 java.sql.ResultSetMetaData.....	192
5.4.4.8 java.sql.Statement.....	193
5.4.4.9 javax.sql.ConnectionPoolDataSource.....	195
5.4.4.10 javax.sql.DataSource.....	196
5.4.4.11 javax.sql.PooledConnection.....	196
5.4.4.12 javax.naming.Context.....	196
5.4.4.13 javax.naming.spi.InitialContextFactory.....	197
5.4.4.14 CopyManager.....	197
5.4.4.15 PGReplicationConnection.....	199
5.4.4.16 PGReplicationStream.....	200
5.4.4.17 ChainedStreamBuilder.....	202
5.4.4.18 ChainedCommonStreamBuilder.....	202
5.5 Development Based on ODBC.....	203
5.5.1 Development Process.....	204
5.5.2 Development Process.....	204
5.5.2.1 Obtaining a Source Code Package, ODBC Packages, and Dependent Libraries.....	204
5.5.2.2 Connecting to a Database.....	206

5.5.2.2.1 Configuring a Data Source in the Linux OS.....	206
5.5.2.2.2 Configuring a Data Source in the Windows OS.....	219
5.5.2.2.3 APIs for Connecting to a Database.....	224
5.5.2.3 Running SQL Statements.....	225
5.5.2.4 Processing Data in a Result Set.....	226
5.5.2.5 Closing a Connection.....	227
5.5.3 Typical Application Development Examples.....	228
5.5.3.1 Typical Application Scenarios and Configurations.....	228
5.5.3.2 Obtaining and Processing Data in a Database.....	229
5.5.3.3 Batch Binding.....	231
5.5.3.4 High-Performance Binding Type.....	237
5.5.4 ODBC Interface Reference.....	244
5.5.4.1 SQLAllocEnv.....	244
5.5.4.2 SQLAllocConnect.....	244
5.5.4.3 SQLAllocHandle.....	244
5.5.4.4 SQLAllocStmt.....	246
5.5.4.5 SQLBindCol.....	246
5.5.4.6 SQLBindParameter.....	247
5.5.4.7 SQLColAttribute.....	249
5.5.4.8 SQLConnect.....	250
5.5.4.9 SQLDisconnect.....	251
5.5.4.10 SQLExecDirect.....	252
5.5.4.11 SQLExecute.....	253
5.5.4.12 SQLFetch.....	254
5.5.4.13 SQLFreeStmt.....	255
5.5.4.14 SQLFreeConnect.....	255
5.5.4.15 SQLFreeHandle.....	255
5.5.4.16 SQLFreeEnv.....	256
5.5.4.17 SQLPrepare.....	256
5.5.4.18 SQLGetData.....	257
5.5.4.19 SQLGetDiagRec.....	259
5.5.4.20 SQLSetConnectAttr.....	261
5.5.4.21 SQLSetEnvAttr.....	262
5.5.4.22 SQLSetStmtAttr.....	263
5.6 Development Based on libpq.....	264
5.6.1 Development Process.....	265
5.6.2 Development Process.....	265
5.6.2.1 Obtaining a Driver Package, a Dependency Library, and Header Files.....	265
5.6.2.2 Connecting to a Database.....	266
5.6.2.3 Running SQL Statements.....	267
5.6.2.4 Processing Data in a Result Set.....	268
5.6.2.5 Closing a Connection.....	269

5.6.3 Typical Application Development Examples.....	269
5.6.3.1 Establishing a Database Connection, Executing SQL Statements, and Returning Results.....	269
5.6.3.2 Executing Prepared Statements.....	272
5.6.3.3 Binding Parameters and Returning a Binary Result.....	273
5.6.4 libpq Interface Reference.....	276
5.6.4.1 Database Connection Control Functions.....	276
5.6.4.1.1 PQconnectdbParams.....	276
5.6.4.1.2 PQconnectdb.....	277
5.6.4.1.3 PQbackendPID.....	278
5.6.4.1.4 PQsetdbLogin.....	278
5.6.4.1.5 PQfinish.....	279
5.6.4.1.6 PQreset.....	280
5.6.4.1.7 PQstatus.....	280
5.6.4.2 Database Statement Execution Functions.....	281
5.6.4.2.1 PQexec.....	281
5.6.4.2.2 PQprepare.....	282
5.6.4.2.3 PQclear.....	284
5.6.4.3 Asynchronous Command Processing Functions.....	284
5.6.4.3.1 PQsendQuery.....	285
5.6.4.3.2 PQsendQueryParams.....	285
5.6.4.3.3 PQsendPrepare.....	286
5.6.4.3.4 PQsendQueryPrepared.....	287
5.6.4.3.5 PQflush.....	288
5.6.4.4 Functions for Canceling Query Processing.....	289
5.6.4.4.1 PQgetCancel.....	289
5.6.4.4.2 PQfreeCancel.....	289
5.6.4.4.3 PQcancel.....	290
5.6.4.5 Functions for Processing Database Result.....	291
5.6.4.5.1 PQgetvalue.....	291
5.6.4.5.2 PQnfields.....	291
5.6.4.5.3 PQntuples.....	292
5.6.4.5.4 PQfname.....	293
5.6.4.5.5 PQresultStatus.....	293
5.7 Psycopg-based Development.....	294
5.7.1 Development Process.....	295
5.7.2 Development Procedure.....	295
5.7.3 Examples: Common Operations.....	299
5.7.4 Psycopg API Reference.....	300
5.7.4.1 psycopg2.connect().....	301
5.7.4.2 connection.cursor().....	303
5.7.4.3 cursor.execute(query,vars_list).....	304
5.7.4.4 cursor.executemany(query,vars_list).....	304

5.7.4.5 connection.commit()	305
5.7.4.6 connection.rollback()	305
5.7.4.7 cursor.fetchone()	306
5.7.4.8 cursor.fetchall()	306
5.7.4.9 cursor.close()	307
5.7.4.10 connection.close()	307
5.8 ECPG-based Development	308
5.8.1 Development Process	309
5.8.2 ecpg Components	310
5.8.3 ecpg Preprocessing and Compiling	311
5.8.4 Managing Database Connections	312
5.8.4.1 Connecting to a Database	312
5.8.4.2 Managing Connections	313
5.8.5 Running SQL Commands	314
5.8.5.1 Running SQL Statements	314
5.8.5.2 Using Cursors	315
5.8.5.3 Transaction	316
5.8.5.4 Prepared Statements	316
5.8.5.5 Embedded SQL Commands	317
5.8.5.5.1 ALLOCATE DESCRIPTOR	317
5.8.5.5.2 CONNECT	317
5.8.5.5.3 DEALLOCATE DESCRIPTOR	320
5.8.5.5.4 DECLARE	320
5.8.5.5.5 DESCRIBE	321
5.8.5.5.6 DISCONNECT	322
5.8.5.5.7 EXECUTE IMMEDIATE	322
5.8.5.5.8 GET DESCRIPTOR	323
5.8.5.5.9 OPEN	325
5.8.5.5.10 PREPARE	325
5.8.5.5.11 SET AUTOCOMMIT	326
5.8.5.5.12 SET CONNECTION	326
5.8.5.5.13 SET DESCRIPTOR	327
5.8.5.5.14 TYPE	328
5.8.5.5.15 VAR	329
5.8.5.5.16 WHENEVER	330
5.8.6 Querying the Result Set	330
5.8.7 Closing a Database Connection	331
5.8.8 Host Variables	331
5.8.8.1 Overview	332
5.8.8.2 DECLARE Section	332
5.8.8.3 Retrieving Query Results	332
5.8.8.4 Type Mapping	333

5.8.8.5 Handling Character Strings.....	334
5.8.8.6 Host Variables with Non-primitive Types.....	335
5.8.8.7 Accessing Special Data Types.....	337
5.8.8.8 Handling Non-primitive SQL Data Types.....	339
5.8.9 Executing Dynamic SQL Statements.....	341
5.8.9.1 Executing a Statement Without a Result Set.....	342
5.8.9.2 Executing a Statement with Input Parameters.....	342
5.8.9.3 Executing a Statement with a Result Set.....	342
5.8.10 Error Handling.....	343
5.8.10.1 Setting Callbacks.....	343
5.8.10.2 sqlca.....	344
5.8.10.3 SQLSTATE and SQLCODE.....	345
5.8.11 Preprocessor Directives.....	349
5.8.11.1 Including Files.....	349
5.8.11.2 Directives: ifdef, ifndef, else, elif, and endif.....	350
5.8.11.3 Directives: define and undef.....	350
5.8.12 Using Library Functions.....	351
5.8.13 SQL Descriptor Area.....	353
5.8.13.1 Named SQLDA.....	353
5.8.13.2 C-Structure SQLDA.....	354
5.8.14 Examples.....	358
5.8.15 ecpg and Pro*C Compatibility Comparison.....	365
5.8.16 ECPG API Reference.....	366
5.8.16.1 Interval Type.....	366
5.8.16.2 Numeric Type.....	367
5.8.16.3 Date Type.....	371
5.8.16.4 Timestamp Type.....	375
5.9 Development Based on the Go Driver.....	378
5.9.1 Setting Up the Go Driver Environment.....	378
5.9.2 Development Process.....	380
5.9.3 Connecting to a Database.....	381
5.9.4 Connecting to a Database (Using SSL).....	390
5.9.5 Go APIs.....	391
5.9.5.1 sql.Open.....	391
5.9.5.2 type DB.....	392
5.9.5.3 type Stmt.....	394
5.9.5.4 type Tx.....	399
5.9.5.5 type Rows.....	401
5.9.5.6 type Row.....	402
5.9.5.7 type ColumnType.....	402
5.9.5.8 type Result.....	403
5.10 Appendix.....	403

5.10.1 JDBC.....	403
5.10.1.1 Data Type Mapping.....	403
5.10.1.2 Log Management.....	404
5.10.1.3 FAQ.....	408
5.10.1.3.1 Incorrect batchMode Settings.....	408
5.10.1.3.2 Error Is Reported When Verification Is Enabled for Data Insertion in the Hibernate Framework.....	409
5.10.1.3.3 Error Is Reported or Connection Is Blocked in SSL Mode.....	411
5.10.2 libpq.....	412
5.10.2.1 Connection Parameters.....	413
5.10.3 Parameters Related to Log Output.....	418
6 SQL Optimization.....	423
6.1 Query Execution Process.....	423
6.2 Introduction to the SQL Execution Plan.....	426
6.2.1 Overview.....	426
6.2.2 Execution Plan Details.....	428
6.2.2.1 Execution Plans.....	428
6.2.2.2 Distributed Plan.....	429
6.2.2.3 Execution Information.....	431
6.2.3 Operators.....	436
6.2.3.1 Keyword Overview.....	436
6.2.3.2 Table Access Modes.....	440
6.2.3.2.1 Seq Scan.....	440
6.2.3.2.2 Index Scan.....	442
6.2.3.2.3 Index Only Scan.....	444
6.2.3.2.4 Bitmap.....	445
6.2.3.2.5 TID Scan.....	448
6.2.3.2.6 Cte Scan.....	449
6.2.3.2.7 Function Scan.....	450
6.2.3.2.8 Sample Scan.....	450
6.2.3.2.9 SubQuery Scan.....	451
6.2.3.2.10 Values Scan.....	452
6.2.3.2.11 WorkTable Scan.....	453
6.2.3.3 Table Join Modes.....	454
6.2.3.3.1 Nested Loop Join.....	454
6.2.3.3.2 Hash Join.....	455
6.2.3.3.3 Merge Join.....	457
6.2.3.4 Operators.....	458
6.2.3.4.1 Sort.....	458
6.2.3.4.2 Limit.....	459
6.2.3.4.3 Append.....	460
6.2.3.4.4 Agg.....	460

6.2.3.4.5 Group.....	462
6.2.3.4.6 MergeAppend.....	463
6.2.3.4.7 SetOp.....	464
6.2.3.4.8 RecursiveUnion.....	467
6.2.3.4.9 Unique.....	468
6.2.3.4.10 LockRows.....	468
6.2.3.4.11 Materialize.....	469
6.2.3.4.12 Result.....	470
6.2.3.4.13 WindowAgg.....	471
6.2.3.4.14 Rownum.....	472
6.2.3.4.15 Unpivot.....	473
6.2.3.5 Distributed Operators.....	474
6.2.3.5.1 Data Node Scan.....	474
6.2.3.5.2 Streaming.....	475
6.2.3.5.3 Remote query.....	476
6.2.3.6 Partition Pruning.....	476
6.2.3.6.1 Partition Iterator.....	476
6.2.3.7 Other Keywords.....	482
6.2.3.7.1 InitPlan and Subplan.....	482
6.3 Optimization Process.....	484
6.4 Updating Statistics.....	484
6.5 Reviewing and Modifying a Table Definition.....	485
6.5.1 Overview.....	486
6.5.2 Selecting a Distribution Mode.....	486
6.5.3 Selecting Distribution Keys.....	488
6.5.4 Using Partitioned Tables.....	488
6.5.5 Selecting a Data Type.....	489
6.6 Reviewing the Plan Change Scenario.....	489
6.7 Typical SQL Optimization Methods.....	490
6.7.1 Optimizing SQL Self-diagnosis.....	490
6.7.2 Optimizing Statement Pushdown.....	493
6.7.3 Optimizing Subqueries.....	501
6.7.4 Optimizing Statistics.....	510
6.7.5 Optimizing Operators.....	515
6.8 Experience in Rewriting SQL Statements.....	517
6.9 Configuring Key Parameters for SQL Tuning.....	519
6.10 Hint-based Tuning.....	521
6.10.1 Plan Hint Optimization.....	521
6.10.2 Hint Specifying the Query Block Where the Hint Is Located.....	527
6.10.3 Hint Specifying the Query Block and Schema of a Table.....	530
6.10.4 Join Order Hints.....	531
6.10.5 Join Operation Hints.....	533

6.10.6 Rows Hints.....	534
6.10.7 Stream Operation Hints.....	535
6.10.8 Scan Hints.....	537
6.10.9 Sublink Name Hints.....	539
6.10.10 Skew Hints.....	540
6.10.11 Parameterized Path Hint.....	542
6.10.12 Hint Errors, Conflicts, and Other Warnings.....	543
6.10.13 GUC Parameter Hints.....	545
6.10.14 Hints for Selecting the Custom Plan or Generic Plan.....	547
6.10.15 Hints Specifying Not to Expand Subqueries.....	548
6.10.16 Hints Specifying Not to Use Global Plan Cache.....	549
6.10.17 Hint of Parameterized Paths at the Same Level.....	550
6.10.18 Hint for Setting Slow SQL Control Rules.....	552
6.10.19 Bitmap Scan Hints.....	552
6.10.20 Hint for Inner Table Materialization During Join.....	553
6.10.21 AGG Hint.....	554
6.11 Checking the Implicit Conversion Performance.....	555
6.12 Tuning with SQL PATCH.....	556
6.13 Optimization Cases.....	562
6.13.1 Case: Selecting an Appropriate Distribution Key.....	562
6.13.2 Case: Creating an Appropriate Index.....	563
6.13.3 Case: Adjusting Distribution Keys.....	565
6.13.4 Case: Adjusting the GUC Parameter best_agg_plan.....	565
6.13.5 Case: Rewriting SQL Statements to Eliminate Subqueries.....	567
6.13.6 Case: Rewriting SQL Statements to Eliminate Pruning Interference.....	568
6.13.7 Case: Rewriting SQL Statements and Deleting in-clause.....	569
6.13.8 Case: Modifying the GUC Parameter rewrite_rule.....	570
6.13.9 Using DN Gather to Reduce Stream Nodes in the Plan.....	577
7 SQL Reference.....	586
7.1 SQL.....	586
7.2 Keywords.....	587
7.3 Data Type.....	624
7.3.1 Numeric Types.....	624
7.3.2 Monetary Types.....	630
7.3.3 Boolean Types.....	631
7.3.4 Character Types.....	632
7.3.5 Binary Types.....	639
7.3.6 Date/Time Types.....	642
7.3.7 Geometric Types.....	652
7.3.8 Network Address Types.....	655
7.3.9 Bit String Types.....	658
7.3.10 UUID Type.....	659

7.3.11 JSON/JSONB Types.....	660
7.3.12 HLL Types.....	664
7.3.13 Range Types.....	668
7.3.14 Object Identifier Types.....	672
7.3.15 Pseudo-Types.....	674
7.3.16 XML Type.....	676
7.3.17 XMLType.....	678
7.3.18 ACLItem.....	679
7.3.19 Array Types.....	680
7.4 Constants and Macros.....	685
7.5 Functions and Operators.....	686
7.5.1 Logical Operators.....	687
7.5.2 Comparison Operators.....	687
7.5.3 Character Processing Functions and Operators.....	688
7.5.4 Binary String Functions and Operators.....	724
7.5.5 Bit String Functions and Operators.....	728
7.5.6 Pattern Matching Operators.....	730
7.5.7 Arithmetic Functions and Operators.....	735
7.5.8 Date and Time Processing Functions and Operators.....	752
7.5.9 Type Conversion Functions.....	784
7.5.10 Geometric Functions and Operators.....	814
7.5.11 Network Address Functions and Operators.....	825
7.5.12 Text Search Functions and Operators.....	830
7.5.13 JSON/JSONB Functions and Operators.....	836
7.5.14 HLL Functions and Operators.....	847
7.5.15 SEQUENCE Functions.....	860
7.5.16 Array Functions and Operators.....	862
7.5.17 Range Functions and Operators.....	871
7.5.18 Aggregate Functions.....	876
7.5.19 Window Functions.....	889
7.5.20 Security Functions.....	895
7.5.21 Encrypted Functions and Operators.....	902
7.5.22 Set Returning Functions.....	906
7.5.23 Conditional Expression Functions.....	908
7.5.24 System Information Functions.....	912
7.5.25 System Administration Functions.....	955
7.5.25.1 Configuration Settings Functions.....	955
7.5.25.2 Universal File Access Functions.....	956
7.5.25.3 Server Signal Functions.....	958
7.5.25.4 Backup and Restoration Control Functions.....	960
7.5.25.5 Dual-Cluster DR Control Functions.....	970
7.5.25.6 Dual-Cluster DR Query Functions.....	970

7.5.25.7 Snapshot Synchronization Functions.....	974
7.5.25.8 Database Object Functions.....	974
7.5.25.9 Advisory Lock Functions.....	980
7.5.25.10 Logical Replication Functions.....	987
7.5.25.11 Undo System Functions.....	1007
7.5.25.12 Other Functions.....	1028
7.5.25.13 SQL Statement Concurrency Control Function.....	1061
7.5.26 Statistics Information Functions.....	1067
7.5.27 Trigger Functions.....	1150
7.5.28 Hash Function.....	1151
7.5.29 Prompt Message Function.....	1159
7.5.30 Fault Injection System Function.....	1159
7.5.31 Redistribution Parameters.....	1160
7.5.32 Distribution Key Recommendation Functions.....	1161
7.5.33 Internal Functions.....	1166
7.5.34 AI Feature Functions.....	1173
7.5.35 Dynamic Data Masking Functions.....	1176
7.5.36 Hotkey Feature Functions.....	1178
7.5.37 Global SysCache Functions.....	1178
7.5.38 Data Damage Detection and Repair Functions.....	1180
7.5.39 Functions of the XML Type.....	1195
7.5.40 Functions of the XMLType Type.....	1211
7.5.41 Other System Functions.....	1219
7.5.42 Obsolete Functions.....	1247
7.6 Expressions.....	1248
7.6.1 Simple Expressions.....	1248
7.6.2 Condition Expressions.....	1250
7.6.3 Subquery Expressions.....	1254
7.6.4 Array Expressions.....	1257
7.6.5 Row Expressions.....	1259
7.7 Pseudocolumn.....	1261
7.8 Type Conversion.....	1264
7.8.1 Overview.....	1264
7.8.2 Operators.....	1266
7.8.3 Functions.....	1268
7.8.4 Value Storage.....	1270
7.8.5 UNION, CASE, and Related Constructs.....	1271
7.9 System Operation.....	1275
7.10 Controlling Transactions.....	1276
7.11 DDL Syntax Overview.....	1277
7.12 DML Syntax Overview.....	1281
7.13 DCL Syntax Overview.....	1283

7.14 SQL Syntax.....	1284
7.14.1 SQL Syntax.....	1284
7.14.2 ABORT.....	1284
7.14.3 ALTER AUDIT POLICY.....	1285
7.14.4 ALTER COLUMN ENCRYPTION KEY.....	1287
7.14.5 ALTER COORDINATOR.....	1288
7.14.6 ALTER DATABASE.....	1289
7.14.7 ALTER DATABASE LINK.....	1292
7.14.8 ALTER DEFAULT PRIVILEGES.....	1293
7.14.9 ALTER DIRECTORY.....	1296
7.14.10 ALTER FOREIGN DATA WRAPPER.....	1297
7.14.11 ALTER FUNCTION.....	1298
7.14.12 ALTER GLOBAL CONFIGURATION.....	1302
7.14.13 ALTER GROUP.....	1303
7.14.14 ALTER INDEX.....	1304
7.14.15 ALTER LANGUAGE.....	1306
7.14.16 ALTER MASKING POLICY.....	1306
7.14.17 ALTER MATERIALIZED VIEW.....	1308
7.14.18 ALTER NODE.....	1309
7.14.19 ALTER NODE GROUP.....	1310
7.14.20 ALTER RESOURCE LABEL.....	1312
7.14.21 ALTER ROLE.....	1313
7.14.22 ALTER ROW LEVEL SECURITY POLICY.....	1315
7.14.23 ALTER SCHEMA.....	1317
7.14.24 ALTER SEQUENCE.....	1319
7.14.25 ALTER SERVER.....	1320
7.14.26 ALTER SESSION.....	1322
7.14.27 ALTER SYNONYM.....	1324
7.14.28 ALTER SYSTEM KILL SESSION.....	1325
7.14.29 ALTER TABLE.....	1326
7.14.30 ALTER TABLE PARTITION.....	1344
7.14.31 ALTER TABLESPACE.....	1351
7.14.32 ALTER TRIGGER.....	1353
7.14.33 ALTER TYPE.....	1354
7.14.34 ALTER USER.....	1356
7.14.35 ALTER VIEW.....	1358
7.14.36 ANALYZE ANALYSE.....	1360
7.14.37 BEGIN.....	1364
7.14.38 CALL.....	1365
7.14.39 CHECKPOINT.....	1366
7.14.40 CLEAN CONNECTION.....	1367
7.14.41 CLOSE.....	1369

7.14.42 CLUSTER.....	1369
7.14.43 COMMENT.....	1372
7.14.44 COMMIT END.....	1374
7.14.45 COMMIT PREPARED.....	1375
7.14.46 COPY.....	1376
7.14.47 CREATE AUDIT POLICY.....	1392
7.14.48 CREATE BARRIER.....	1394
7.14.49 CREATE CLIENT MASTER KEY.....	1395
7.14.50 CREATE COLUMN ENCRYPTION KEY.....	1396
7.14.51 CREATE CONVERSION.....	1398
7.14.52 CREATE DATABASE.....	1399
7.14.53 CREATE DATABASE LINK.....	1408
7.14.54 CREATE DIRECTORY.....	1409
7.14.55 CREATE FOREIGN DATA WRAPPER.....	1411
7.14.56 CREATE FUNCTION.....	1412
7.14.57 CREATE GLOBAL INDEX.....	1422
7.14.58 CREATE GROUP.....	1426
7.14.59 CREATE INCREMENTAL MATERIALIZED VIEW.....	1427
7.14.60 CREATE INDEX.....	1428
7.14.61 CREATE LANGUAGE.....	1438
7.14.62 CREATE MASKING POLICY.....	1438
7.14.63 CREATE MATERIALIZED VIEW.....	1440
7.14.64 CREATE MODEL.....	1442
7.14.65 CREATE NODE.....	1442
7.14.66 CREATE NODE GROUP.....	1444
7.14.67 CREATE PROCEDURE.....	1445
7.14.68 CREATE RESOURCE LABEL.....	1449
7.14.69 CREATE ROLE.....	1451
7.14.70 CREATE ROW LEVEL SECURITY POLICY.....	1455
7.14.71 CREATE SCHEMA.....	1459
7.14.72 CREATE SEQUENCE.....	1461
7.14.73 CREATE SERVER.....	1464
7.14.74 CREATE SYNONYM.....	1465
7.14.75 CREATE TABLE.....	1468
7.14.76 CREATE TABLESPACE.....	1492
7.14.77 CREATE TABLE AS.....	1494
7.14.78 CREATE TABLE PARTITION.....	1497
7.14.79 CREATE TRIGGER.....	1516
7.14.80 CREATE TYPE.....	1521
7.14.81 CREATE USER.....	1529
7.14.82 CREATE VIEW.....	1531
7.14.83 CREATE WEAK PASSWORD DICTIONARY.....	1532

7.14.84 CURSOR.....	1533
7.14.85 DEALLOCATE.....	1535
7.14.86 DECLARE.....	1535
7.14.87 DELETE.....	1537
7.14.88 DO.....	1540
7.14.89 DROP AUDIT POLICY.....	1541
7.14.90 DROP CLIENT MASTER KEY.....	1541
7.14.91 DROP COLUMN ENCRYPTION KEY.....	1542
7.14.92 DROP DATABASE.....	1543
7.14.93 DROP DATABASE LINK.....	1544
7.14.94 DROP DIRECTORY.....	1544
7.14.95 DROP FOREIGN DATA WRAPPER.....	1545
7.14.96 DROP FUNCTION.....	1546
7.14.97 DROP GLOBAL CONFIGURATION.....	1547
7.14.98 DROP GROUP.....	1547
7.14.99 DROP INDEX.....	1548
7.14.100 DROP LANGUAGE.....	1548
7.14.101 DROP MASKING POLICY.....	1548
7.14.102 DROP MATERIALIZED VIEW.....	1549
7.14.103 DROP MODEL.....	1550
7.14.104 DROP NODE.....	1550
7.14.105 DROP NODE GROUP.....	1551
7.14.106 DROP OWNED.....	1551
7.14.107 DROP PROCEDURE.....	1552
7.14.108 DROP RESOURCE LABEL.....	1553
7.14.109 DROP ROLE.....	1553
7.14.110 DROP ROW LEVEL SECURITY POLICY.....	1554
7.14.111 DROP SCHEMA.....	1555
7.14.112 DROP SEQUENCE.....	1556
7.14.113 DROP SERVER.....	1556
7.14.114 DROP SYNONYM.....	1557
7.14.115 DROP TABLE.....	1558
7.14.116 DROP TABLESPACE.....	1559
7.14.117 DROP TRIGGER.....	1560
7.14.118 DROP TYPE.....	1561
7.14.119 DROP USER.....	1561
7.14.120 DROP VIEW.....	1562
7.14.121 DROP WEAK PASSWORD DICTIONARY.....	1563
7.14.122 EXECUTE.....	1564
7.14.123 EXECUTE DIRECT.....	1565
7.14.124 EXPDP DATABASE.....	1566
7.14.125 EXPDP TABLE.....	1567

7.14.126 EXPLAIN.....	1567
7.14.127 EXPLAIN PLAN.....	1572
7.14.128 FETCH.....	1574
7.14.129 GRANT.....	1578
7.14.130 IMPDP DATABASE CREATE.....	1590
7.14.131 IMPDP RECOVER.....	1590
7.14.132 INSERT.....	1591
7.14.133 IMPDP TABLE.....	1595
7.14.134 IMPDP TABLE PREPARE.....	1596
7.14.135 LOCK.....	1596
7.14.136 MOVE.....	1600
7.14.137 MERGE INTO.....	1602
7.14.138 PREDICT BY.....	1604
7.14.139 PREPARE.....	1604
7.14.140 PREPARE TRANSACTION.....	1605
7.14.141 PURGE.....	1606
7.14.142 REASSIGN OWNED.....	1608
7.14.143 REINDEX.....	1609
7.14.144 REFRESH INCREMENTAL MATERIALIZED VIEW.....	1612
7.14.145 REFRESH MATERIALIZED VIEW.....	1612
7.14.146 RELEASE SAVEPOINT.....	1613
7.14.147 RESET.....	1615
7.14.148 REVOKE.....	1616
7.14.149 ROLLBACK.....	1619
7.14.150 ROLLBACK PREPARED.....	1620
7.14.151 ROLLBACK TO SAVEPOINT.....	1621
7.14.152 SAVEPOINT.....	1622
7.14.153 SELECT.....	1624
7.14.154 SELECT INTO.....	1643
7.14.155 SET.....	1644
7.14.156 SET CONSTRAINTS.....	1646
7.14.157 SET ROLE.....	1647
7.14.158 SET SESSION AUTHORIZATION.....	1648
7.14.159 SET TRANSACTION.....	1650
7.14.160 SHOW.....	1651
7.14.161 SHUTDOWN.....	1651
7.14.162 START TRANSACTION.....	1652
7.14.163 TIMECAPSULE TABLE.....	1654
7.14.164 TRUNCATE.....	1657
7.14.165 UPDATE.....	1660
7.14.166 VACUUM.....	1664
7.14.167 VALUES.....	1667

7.15 Appendix.....	1669
7.15.1 Extended Functions.....	1669
7.15.2 Dollar-Quoted String Constants.....	1669
7.15.3 DATABASE LINK.....	1669
8 Best Practices.....	1681
8.1 Best Practices of Table Design.....	1681
8.1.1 Selecting a Distribution Mode.....	1681
8.1.2 Selecting Distribution Keys.....	1682
8.1.3 Using Partitioned Tables.....	1683
8.1.4 Selecting a Data Type.....	1683
8.1.5 Checking a Node Where a Table Resides.....	1684
8.2 Best Practices of SQL Queries.....	1684
8.3 Best Practices for Data Skew Query.....	1686
8.3.1 Quickly Locating Tables That Cause Data Skew.....	1687
9 User-defined Functions.....	1689
9.1 PL/SQL Functions.....	1689
10 Stored Procedures.....	1691
10.1 Overview.....	1691
10.2 Data Types.....	1691
10.3 Data Type Conversion.....	1691
10.4 DECLARE Syntax.....	1693
10.4.1 Basic Structure.....	1693
10.4.2 Anonymous Blocks.....	1693
10.4.3 Subprograms.....	1695
10.5 Basic Statements.....	1695
10.5.1 Variable Definition Statements.....	1695
10.5.2 Assignment Statements.....	1697
10.5.3 Call Statements.....	1699
10.6 Dynamic Statements.....	1700
10.6.1 Executing Dynamic Query Statements.....	1700
10.6.2 Executing Dynamic Non-Query Statements.....	1704
10.6.3 Dynamically Calling Stored Procedures.....	1705
10.6.4 Dynamically Calling Anonymous Blocks.....	1707
10.7 Control Statements.....	1709
10.7.1 RETURN Statements.....	1709
10.7.1.1 RETURN.....	1709
10.7.1.2 RETURN NEXT and RETURN QUERY.....	1710
10.7.2 Conditional Statements.....	1711
10.7.3 Loop Statements.....	1713
10.7.4 Branch Statements.....	1717
10.7.5 NULL Statements.....	1719

10.7.6 Error Trapping Statements.....	1719
10.7.7 GOTO Statements.....	1722
10.8 Transaction Statements.....	1723
10.9 Other Statements.....	1732
10.9.1 Lock Operations.....	1732
10.9.2 Cursor Operations.....	1732
10.10 Cursors.....	1733
10.10.1 Overview.....	1733
10.10.2 Explicit Cursor.....	1733
10.10.3 Implicit Cursor.....	1739
10.10.4 Cursor Loop.....	1740
10.11 Advanced Packages.....	1741
10.11.1 Basic APIs.....	1741
10.11.1.1 PKG_SERVICE.....	1741
10.11.1.2 PKG_UTIL.....	1752
10.11.1.3 DBE_XML.....	1798
10.11.2 Secondary Encapsulation APIs (Recommended).....	1823
10.11.2.1 DBE_LOB.....	1823
10.11.2.2 DBE_RANDOM.....	1851
10.11.2.3 DBE_OUTPUT.....	1852
10.11.2.4 DBE_RAW.....	1858
10.11.2.5 DBE_TASK.....	1872
10.11.2.6 DBE_UTILITY.....	1884
10.11.2.7 DBE_SQL.....	1900
10.11.2.8 DBE_FILE.....	1927
10.11.2.9 DBE_SESSION.....	1955
10.11.2.10 DBE_MATCH.....	1958
10.11.2.11 DBE_SCHEDULER.....	1959
10.11.2.12 DBE_APPLICATION_INFO.....	1985
10.11.2.13 DBE_XMLDOM.....	1986
10.11.2.14 DBE_XMLPARSER.....	2024
10.12 Retry Management.....	2031
10.13 Debugging.....	2032
11 Autonomous Transaction.....	2035
11.1 Stored Procedure Supporting Autonomous Transaction.....	2035
11.2 Anonymous Block Supporting Autonomous Transaction.....	2036
11.3 Function Supporting Autonomous Transaction.....	2037
11.4 Restrictions.....	2038
12 System Catalogs and System Views.....	2042
12.1 Overview of System Catalogs and System Views.....	2042
12.2 System Catalogs.....	2042
12.2.1 GS_AUDITING_POLICY.....	2043

12.2.2 GS_AUDITING_POLICY_ACCESS.....	2043
12.2.3 GS_AUDITING_POLICY_FILTERS.....	2044
12.2.4 GS_AUDITING_POLICY_PRIVILEGES.....	2044
12.2.5 GS_ASP.....	2045
12.2.6 GS_CLIENT_GLOBAL_KEYS.....	2047
12.2.7 GS_CLIENT_GLOBAL_KEYS_ARGS.....	2048
12.2.8 GS_COLUMN_KEYS.....	2048
12.2.9 GS_COLUMN_KEYS_ARGS.....	2049
12.2.10 GS_DATABASE_LINK.....	2049
12.2.11 GS_DB_PRIVILEGE.....	2050
12.2.12 GS_DEPENDENCIES.....	2050
12.2.13 GS_DEPENDENCIES_OBJ.....	2051
12.2.14 GS_ENCRYPTED_COLUMNS.....	2051
12.2.15 GS_ENCRYPTED_PROC.....	2052
12.2.16 GS_GLOBAL_CONFIG.....	2053
12.2.17 GS_JOB_ATTRIBUTE.....	2053
12.2.18 GS_JOB_ARGUMENT.....	2054
12.2.19 GS_MASKING_POLICY.....	2054
12.2.20 GS_MASKING_POLICY_ACTIONS.....	2055
12.2.21 GS_MASKING_POLICY_FILTERS.....	2056
12.2.22 GS_MATVIEW.....	2056
12.2.23 GS_MATVIEW_DEPENDENCY.....	2057
12.2.24 GS_MODEL_WAREHOUSE.....	2057
12.2.25 GS_OPT_MODEL.....	2059
12.2.26 GS_PLAN_TRACE.....	2059
12.2.27 GS_POLICY_LABEL.....	2060
12.2.28 GS_RECYCLEBIN.....	2060
12.2.29 GS_SQL_PATCH.....	2062
12.2.30 GS_TXN_SNAPSHOT.....	2063
12.2.31 GS_UID.....	2063
12.2.32 GS_WORKLOAD_RULE.....	2064
12.2.33 PG_AGGREGATE.....	2065
12.2.34 PG_AM.....	2066
12.2.35 PG_AMOP.....	2069
12.2.36 PG_AMPROC.....	2070
12.2.37 PG_APP_WORKLOADGROUP_MAPPING.....	2071
12.2.38 PG_ATTRDEF.....	2071
12.2.39 PG_ATTRIBUTE.....	2072
12.2.40 PG_AUTHID.....	2074
12.2.41 PG_AUTH_HISTORY.....	2077
12.2.42 PG_AUTH_MEMBERS.....	2077
12.2.43 PG_CAST.....	2078

12.2.44 PG_CLASS.....	2078
12.2.45 PG_COLLATION.....	2083
12.2.46 PG_CONSTRAINT.....	2084
12.2.47 PG_CONVERSION.....	2087
12.2.48 PG_DATABASE.....	2088
12.2.49 PG_DB_ROLE_SETTING.....	2089
12.2.50 PG_DEFAULT_ACL.....	2090
12.2.51 PG_DEPEND.....	2090
12.2.52 PG_DESCRIPTION.....	2092
12.2.53 PG_DIRECTORY.....	2092
12.2.54 PG_ENUM.....	2093
12.2.55 PG_EXTENSION.....	2093
12.2.56 PG_FOREIGN_DATA_WRAPPER.....	2094
12.2.57 PG_FOREIGN_SERVER.....	2095
12.2.58 PG_HASHBUCKET.....	2096
12.2.59 PG_INDEX.....	2096
12.2.60 PG_INHERITS.....	2098
12.2.61 PG_JOB.....	2099
12.2.62 PG_JOB_PROC.....	2101
12.2.63 PG_LANGUAGE.....	2101
12.2.64 PG_LARGEOBJECT.....	2103
12.2.65 PG_LARGEOBJECT_METADATA.....	2103
12.2.66 PG_NAMESPACE.....	2104
12.2.67 PG_OBJECT.....	2104
12.2.68 PG_OPCLASS.....	2105
12.2.69 PG_OPERATOR.....	2106
12.2.70 PG_OPFAMILY.....	2107
12.2.71 PG_PARTITION.....	2108
12.2.72 PG_PLTEMPLATE.....	2110
12.2.73 PG_PROC.....	2111
12.2.74 PG_RANGE.....	2115
12.2.75 PG_REPLICATION_ORIGIN.....	2116
12.2.76 PG_RESOURCE_POOL.....	2116
12.2.77 PG_REWRITE.....	2117
12.2.78 PG_RLSPOLICY.....	2118
12.2.79 PG_SECLABEL.....	2119
12.2.80 PG_SET.....	2119
12.2.81 PG_SHDEPEND.....	2119
12.2.82 PG_SHDESCRIPTION.....	2121
12.2.83 PG_SHSECLABEL.....	2121
12.2.84 PG_STATISTIC.....	2122
12.2.85 PG_STATISTIC_EXT.....	2123

12.2.86 PG_SYNONYM.....	2125
12.2.87 PG_TABLESPACE.....	2126
12.2.88 PG_TRIGGER.....	2126
12.2.89 PG_TS_CONFIG.....	2128
12.2.90 PG_TS_CONFIG_MAP.....	2128
12.2.91 PG_TS_DICT.....	2129
12.2.92 PG_TS_PARSER.....	2129
12.2.93 PG_TS_TEMPLATE.....	2130
12.2.94 PG_TYPE.....	2130
12.2.95 PG_USER_MAPPING.....	2134
12.2.96 PG_USER_STATUS.....	2135
12.2.97 PGXC_CLASS.....	2135
12.2.98 PGXC_GROUP.....	2136
12.2.99 PGXC_NODE.....	2137
12.2.100 PGXC_REDISTB.....	2139
12.2.101 PGXC_SLICE.....	2140
12.2.102 PLAN_TABLE_DATA.....	2141
12.2.103 STATEMENT_HISTORY.....	2143
12.2.104 STREAMING_STREAM.....	2149
12.2.105 STREAMING_CONT_QUERY.....	2149
12.3 System Views.....	2150
12.3.1 ADM_ARGUMENTS.....	2150
12.3.2 ADM_AUDIT_OBJECT.....	2152
12.3.3 ADM_AUDIT_SESSION.....	2154
12.3.4 ADM_AUDIT_STATEMENT.....	2156
12.3.5 ADM_AUDIT_TRAIL.....	2158
12.3.6 ADM_COL_COMMENTS.....	2160
12.3.7 ADM_COL_PRIVS.....	2161
12.3.8 ADM_COLL_TYPES.....	2162
12.3.9 ADM_CONS_COLUMNS.....	2163
12.3.10 ADM_CONSTRAINTS.....	2163
12.3.11 ADM_DATA_FILES.....	2164
12.3.12 ADM_DEPENDENCIES.....	2165
12.3.13 ADM_DIRECTORIES.....	2165
12.3.14 ADM_HIST_SNAPSHOT.....	2166
12.3.15 ADM_HIST_SQL_PLAN.....	2167
12.3.16 ADM_HIST_SQLSTAT.....	2169
12.3.17 ADM_HIST_SQLTEXT.....	2170
12.3.18 ADM_IND_COLUMNS.....	2171
12.3.19 ADM_IND_EXPRESSIONS.....	2171
12.3.20 ADM_IND_PARTITIONS.....	2172
12.3.21 ADM_IND_SUBPARTITIONS.....	2176

12.3.22	ADM_INDEXES.....	2178
12.3.23	ADM_OBJECTS.....	2183
12.3.24	ADM_PART_COL_STATISTICS.....	2184
12.3.25	ADM_PART_INDEXES.....	2185
12.3.26	ADM_PART_TABLES.....	2186
12.3.27	ADM_PROCEDURES.....	2189
12.3.28	ADM_ROLE_PRIVS.....	2191
12.3.29	ADM_ROLES.....	2192
12.3.30	ADM_SCHEDULER_JOB_ARGS.....	2192
12.3.31	ADM_SCHEDULER_JOBS.....	2193
12.3.32	ADM_SCHEDULER_PROGRAM_ARGS.....	2197
12.3.33	ADM_SCHEDULER_PROGRAMS.....	2198
12.3.34	ADM_SCHEDULER_RUNNING_JOBS.....	2200
12.3.35	ADM_SEGMENTS.....	2201
12.3.36	ADM_SEQUENCES.....	2203
12.3.37	ADM_SOURCE.....	2203
12.3.38	ADM_SUBPART_KEY_COLUMNS.....	2204
12.3.39	ADM_SYNONYMS.....	2205
12.3.40	ADM_SYS_PRIVS.....	2205
12.3.41	ADM_TAB_COLS.....	2206
12.3.42	ADM_TAB_COL_STATISTICS.....	2209
12.3.43	ADM_TAB_COLUMNS.....	2211
12.3.44	ADM_TAB_COMMENTS.....	2213
12.3.45	ADM_TAB_HISTOGRAMS.....	2213
12.3.46	ADM_TAB_PARTITIONS.....	2214
12.3.47	ADM_TAB_PRIVS.....	2217
12.3.48	ADM_TAB_STATISTICS.....	2218
12.3.49	ADM_TAB_STATS_HISTORY.....	2220
12.3.50	ADM_TABLES.....	2220
12.3.51	ADM_TABLESPACES.....	2227
12.3.52	ADM_TRIGGERS.....	2229
12.3.53	ADM_TYPES.....	2231
12.3.54	ADM_TYPE_ATTRS.....	2232
12.3.55	ADM_USERS.....	2233
12.3.56	ADM_VIEWS.....	2235
12.3.57	COMM_CLIENT_INFO.....	2237
12.3.58	DB_ALL_TABLES.....	2237
12.3.59	DB_ARGUMENTS.....	2238
12.3.60	DB_COL_COMMENTS.....	2240
12.3.61	DB_COL_PRIVS.....	2240
12.3.62	DB_COLL_TYPES.....	2241
12.3.63	DB_CONSTRAINTS.....	2242

12.3.64 DB_CONS_COLUMNS.....	2243
12.3.65 DB_DEPENDENCIES.....	2243
12.3.66 DB_IND_COLUMNS.....	2244
12.3.67 DB_IND_EXPRESSIONS.....	2244
12.3.68 DB_IND_PARTITIONS.....	2245
12.3.69 DB_IND_SUBPARTITIONS.....	2248
12.3.70 DB_INDEXES.....	2251
12.3.71 DB_OBJECTS.....	2255
12.3.72 DB_PART_COL_STATISTICS.....	2257
12.3.73 DB_PART_KEY_COLUMNS.....	2258
12.3.74 DB_PART_TABLES.....	2259
12.3.75 DB_PROCEDURES.....	2262
12.3.76 DB_SCHEDULER_JOB_ARGS.....	2262
12.3.77 DB_SCHEDULER_PROGRAM_ARGS.....	2263
12.3.78 DB_SEQUENCES.....	2263
12.3.79 DB_SOURCE.....	2264
12.3.80 DB_SUBPART_KEY_COLUMNS.....	2265
12.3.81 DB_SYNONYMS.....	2265
12.3.82 DB_TAB_COL_STATISTICS.....	2266
12.3.83 DB_TAB_COLUMNS.....	2268
12.3.84 DB_TAB_COMMENTS.....	2270
12.3.85 DB_TAB_HISTOGRAMS.....	2271
12.3.86 DB_TAB_PARTITIONS.....	2271
12.3.87 DB_TAB_STATS_HISTORY.....	2274
12.3.88 DB_TAB_SUBPARTITIONS.....	2275
12.3.89 DB_TABLES.....	2277
12.3.90 DB_TRIGGERS.....	2284
12.3.91 DB_TYPES.....	2284
12.3.92 DB_USERS.....	2285
12.3.93 DB_VIEWS.....	2285
12.3.94 DICT.....	2287
12.3.95 DICTIONARY.....	2287
12.3.96 DV_SESSIONS.....	2288
12.3.97 DV_SESSION_LONGOPS.....	2288
12.3.98 GET_GLOBAL_PREPARED_XACTS.....	2289
12.3.99 GLOBAL_BAD_BLOCK_INFO.....	2289
12.3.100 GLOBAL_CLEAR_BAD_BLOCK_INFO.....	2290
12.3.101 GLOBAL_COMM_CLIENT_INFO.....	2291
12.3.102 GLOBAL_SQL_PATCH.....	2291
12.3.103 GLOBAL_STAT_HOTKEYS_INFO.....	2292
12.3.104 GLOBAL_WAL_SENDER_STATUS.....	2293
12.3.105 GS_ALL_CONTROL_GROUP_INFO.....	2296

12.3.106 GS_ALL_PREPARED_STATEMENTS.....	2297
12.3.107 GS_AUDITING.....	2298
12.3.108 GS_AUDITING_ACCESS.....	2298
12.3.109 GS_AUDITING_PRIVILEGE.....	2299
12.3.110 GS_CLUSTER_RESOURCE_INFO.....	2299
12.3.111 GS_COMM_LISTEN_ADDRESS_EXT_INFO.....	2300
12.3.112 GS_DB_LINKS.....	2301
12.3.113 GS_DB_PRIVILEGES.....	2301
12.3.114 GS_GET_CONTROL_GROUP_INFO.....	2302
12.3.115 GS_GET_LISTEN_ADDRESS_EXT_INFO.....	2303
12.3.116 GS_GLOBAL_ARCHIVE_STATUS.....	2303
12.3.117 GS_GSC_MEMORY_DETAIL.....	2304
12.3.118 GS_LABELS.....	2304
12.3.119 GS_LSC_MEMORY_DETAIL.....	2305
12.3.120 GS_MASKING.....	2305
12.3.121 GS_MATVIEWS.....	2306
12.3.122 GS_MY_PLAN_TRACE.....	2306
12.3.123 GS_SESSION_ALL_SETTINGS.....	2307
12.3.124 GS_SQL_COUNT.....	2307
12.3.125 GS_STAT_ALL_PARTITIONS.....	2309
12.3.126 GS_STAT_XACT_ALL_PARTITIONS.....	2311
12.3.127 GS_STATIO_ALL_PARTITIONS.....	2311
12.3.128 GS_WORKLOAD_RULE_STAT.....	2312
12.3.129 GV_INSTANCE.....	2313
12.3.130 GV_SESSION.....	2315
12.3.131 MPP_TABLES.....	2321
12.3.132 MY_COL_COMMENTS.....	2321
12.3.133 MY_COL_PRIVS.....	2322
12.3.134 MY_COLL_TYPES.....	2322
12.3.135 MY_CONS_COLUMNS.....	2323
12.3.136 MY_CONSTRAINTS.....	2324
12.3.137 MY_DEPENDENCIES.....	2324
12.3.138 MY_IND_COLUMNS.....	2325
12.3.139 MY_IND_EXPRESSIONS.....	2326
12.3.140 MY_IND_PARTITIONS.....	2326
12.3.141 MY_IND_SUBPARTITIONS.....	2330
12.3.142 MY_INDEXES.....	2333
12.3.143 MY_JOBS.....	2337
12.3.144 MY_OBJECTS.....	2339
12.3.145 MY_PART_COL_STATISTICS.....	2340
12.3.146 MY_PART_INDEXES.....	2341
12.3.147 MY_PART_KEY_COLUMNS.....	2342

12.3.148 MY_PART_TABLES.....	2343
12.3.149 MY_PROCEDURES.....	2346
12.3.150 MY_ROLE_PRIVS.....	2347
12.3.151 MY_SCHEDULER_JOB_ARGS.....	2348
12.3.152 MY_SCHEDULER_PROGRAM_ARGS.....	2349
12.3.153 MY_SEQUENCES.....	2349
12.3.154 MY_SOURCE.....	2351
12.3.155 MY_SUBPART_KEY_COLUMNS.....	2351
12.3.156 MY_SYNONYMS.....	2352
12.3.157 MY_SYS_PRIVS.....	2353
12.3.158 MY_TAB_COL_STATISTICS.....	2353
12.3.159 MY_TAB_COLUMNS.....	2355
12.3.160 MY_TAB_COMMENTS.....	2357
12.3.161 MY_TAB_HISTOGRAMS.....	2358
12.3.162 MY_TAB_PARTITIONS.....	2358
12.3.163 MY_TAB_STATISTICS.....	2361
12.3.164 MY_TAB_STATS_HISTORY.....	2363
12.3.165 MY_TABLES.....	2363
12.3.166 MY_TABLESPACES.....	2367
12.3.167 MY_TRIGGERS.....	2370
12.3.168 MY_TYPE_ATTRS.....	2371
12.3.169 MY_TYPES.....	2372
12.3.170 MY_VIEWS.....	2373
12.3.171 NLS_DATABASE_PARAMETERS.....	2374
12.3.172 NLS_INSTANCE_PARAMETERS.....	2375
12.3.173 PG_AVAILABLE_EXTENSION_VERSIONS.....	2375
12.3.174 PG_AVAILABLE_EXTENSIONS.....	2376
12.3.175 PG_COMM_DELAY.....	2376
12.3.176 PG_COMM_RECV_STREAM.....	2377
12.3.177 PG_COMM_SEND_STREAM.....	2378
12.3.178 PG_COMM_STATUS.....	2379
12.3.179 PG_CONTROL_GROUP_CONFIG.....	2380
12.3.180 PG_CURSORS.....	2380
12.3.181 PG_EXT_STATS.....	2381
12.3.182 PG_GET_INVALID_BACKENDS.....	2383
12.3.183 PG_GET_SENDERS_CATCHUP_TIME.....	2384
12.3.184 PG_GROUP.....	2385
12.3.185 PG_INDEXES.....	2385
12.3.186 PG_LOCKS.....	2386
12.3.187 PG_NODE_ENV.....	2387
12.3.188 PG_OS_THREADS.....	2388
12.3.189 PG_POOLER_STATUS.....	2388

12.3.190 PG_PREPARED_STATEMENTS.....	2389
12.3.191 PG_PREPARED_XACTS.....	2390
12.3.192 PG_REPLICATION_SLOTS.....	2391
12.3.193 PG_RLSPOLICIES.....	2392
12.3.194 PG_ROLES.....	2393
12.3.195 PG_RULES.....	2395
12.3.196 PG_RUNNING_XACTS.....	2396
12.3.197 PG_SECLABELS.....	2396
12.3.198 PG_SETTINGS.....	2397
12.3.199 PG_SHADOW.....	2398
12.3.200 PG_SHARED_MEMORY_DETAIL.....	2400
12.3.201 PG_STAT_ACTIVITY.....	2401
12.3.202 PG_STAT_ALL_INDEXES.....	2404
12.3.203 PG_STAT_ALL_TABLES.....	2405
12.3.204 PG_STAT_BAD_BLOCK.....	2406
12.3.205 PG_STAT_BGWRITER.....	2407
12.3.206 PG_STAT_DATABASE.....	2408
12.3.207 PG_STAT_DATABASE_CONFLICTS.....	2410
12.3.208 PG_STAT_REPLICATION.....	2410
12.3.209 PG_STAT_SYS_INDEXES.....	2412
12.3.210 PG_STAT_SYS_TABLES.....	2413
12.3.211 PG_STAT_USER_FUNCTIONS.....	2414
12.3.212 PG_STAT_USER_INDEXES.....	2415
12.3.213 PG_STAT_USER_TABLES.....	2415
12.3.214 PG_STAT_XACT_ALL_TABLES.....	2417
12.3.215 PG_STAT_XACT_SYS_TABLES.....	2417
12.3.216 PG_STAT_XACT_USER_FUNCTIONS.....	2418
12.3.217 PG_STAT_XACT_USER_TABLES.....	2419
12.3.218 PG_STATS.....	2419
12.3.219 PG_STATIO_ALL_INDEXES.....	2422
12.3.220 PG_STATIO_ALL_SEQUENCES.....	2422
12.3.221 PG_STATIO_ALL_TABLES.....	2422
12.3.222 PG_STATIO_SYS_INDEXES.....	2423
12.3.223 PG_STATIO_SYS_SEQUENCES.....	2424
12.3.224 PG_STATIO_SYS_TABLES.....	2424
12.3.225 PG_STATIO_USER_INDEXES.....	2425
12.3.226 PG_STATIO_USER_SEQUENCES.....	2425
12.3.227 PG_STATIO_USER_TABLES.....	2426
12.3.228 PG_THREAD_WAIT_STATUS.....	2426
12.3.229 PG_TABLES.....	2446
12.3.230 PG_TDE_INFO.....	2447
12.3.231 PG_TIMEZONE_ABBREVS.....	2448

12.3.232 PG_TIMEZONE_NAMES.....	2448
12.3.233 PG_TOTAL_MEMORY_DETAIL.....	2449
12.3.234 PG_TOTAL_USER_RESOURCE_INFO.....	2449
12.3.235 PG_TOTAL_USER_RESOURCE_INFO_OID.....	2451
12.3.236 PG_USER.....	2452
12.3.237 PG_USER_MAPPINGS.....	2453
12.3.238 PG_VARIABLE_INFO.....	2454
12.3.239 PG_VIEWS.....	2454
12.3.240 PGXC_COMM_DELAY.....	2455
12.3.241 PGXC_COMM_RECV_STREAM.....	2455
12.3.242 PGXC_COMM_SEND_STREAM.....	2457
12.3.243 PGXC_COMM_STATUS.....	2458
12.3.244 PGXC_GET_STAT_ALL_TABLES.....	2459
12.3.245 PGXC_GET_TABLE_SKEWNESS.....	2459
12.3.246 PGXC_NODE_ENV.....	2460
12.3.247 PGXC_OS_THREADS.....	2460
12.3.248 PGXC_PREPARED_XACTS.....	2461
12.3.249 PGXC_RUNNING_XACTS.....	2461
12.3.250 PGXC_SQL_COUNT.....	2462
12.3.251 PGXC_STAT_ACTIVITY.....	2463
12.3.252 PGXC_STAT_BAD_BLOCK.....	2466
12.3.253 PGXC_THREAD_WAIT_STATUS.....	2467
12.3.254 PGXC_TOTAL_MEMORY_DETAIL.....	2468
12.3.255 PGXC_VARIABLE_INFO.....	2469
12.3.256 PLAN_TABLE.....	2470
12.3.257 PV_FILE_STAT.....	2472
12.3.258 PV_INSTANCE_TIME.....	2473
12.3.259 PV_OS_RUN_INFO.....	2474
12.3.260 PV_REDO_STAT.....	2474
12.3.261 PV_SESSION_MEMORY.....	2475
12.3.262 PV_SESSION_MEMORY_CONTEXT.....	2475
12.3.263 PV_SESSION_MEMORY_DETAIL.....	2476
12.3.264 PV_SESSION_STAT.....	2477
12.3.265 PV_SESSION_TIME.....	2478
12.3.266 PV_THREAD_MEMORY_CONTEXT.....	2478
12.3.267 PV_TOTAL_MEMORY_DETAIL.....	2479
12.3.268 ROLE_ROLE_PRIVS.....	2480
12.3.269 ROLE_SYS_PRIVS.....	2481
12.3.270 ROLE_TAB_PRIVS.....	2482
12.3.271 SYS_DUMMY.....	2482
12.3.272 V_INSTANCE.....	2483
12.3.273 V_MYSTAT.....	2484

12.3.274 V_SESSION.....	2485
12.3.275 V\$NLS_PARAMETERS.....	2490
12.3.276 V\$SESSION_WAIT.....	2491
12.3.277 V\$SYSSTAT.....	2493
12.3.278 V\$SYSTEM_EVENT.....	2493
12.3.279 V\$VERSION.....	2494
13 Schemas.....	2495
13.1 Information Schema.....	2497
13.1.1 _PG_FOREIGN_DATA_WRAPPERS.....	2497
13.1.2 _PG_FOREIGN_SERVERS.....	2498
13.1.3 _PG_FOREIGN_TABLE_COLUMNS.....	2499
13.1.4 _PG_FOREIGN_TABLES.....	2499
13.1.5 _PG_USER_MAPPINGS.....	2500
13.1.6 INFORMATION_SCHEMA_CATALOG_NAME.....	2500
13.2 DBE_PERF Schema.....	2501
13.2.1 OS.....	2501
13.2.1.1 OS_RUNTIME.....	2501
13.2.1.2 GLOBAL_OS_RUNTIME.....	2502
13.2.1.3 OS_THREADS.....	2502
13.2.1.4 GLOBAL_OS_THREADS.....	2502
13.2.2 Instance.....	2503
13.2.2.1 INSTANCE_TIME.....	2503
13.2.2.2 GLOBAL_INSTANCE_TIME.....	2504
13.2.3 Memory.....	2504
13.2.3.1 MEMORY_NODE_DETAIL.....	2504
13.2.3.2 GLOBAL_MEMORY_NODE_DETAIL.....	2505
13.2.3.3 MEMORY_NODE_NG_DETAIL.....	2507
13.2.3.4 SHARED_MEMORY_DETAIL.....	2507
13.2.3.5 GLOBAL_SHARED_MEMORY_DETAIL.....	2508
13.2.3.6 TRACK_MEMORY_CONTEXT_DETAIL.....	2508
13.2.4 File.....	2509
13.2.4.1 FILE_IOSTAT.....	2509
13.2.4.2 SUMMARY_FILE_IOSTAT.....	2510
13.2.4.3 GLOBAL_FILE_IOSTAT.....	2510
13.2.4.4 FILE_REDO_IOSTAT.....	2511
13.2.4.5 SUMMARY_FILE_REDO_IOSTAT.....	2512
13.2.4.6 GLOBAL_FILE_REDO_IOSTAT.....	2512
13.2.4.7 LOCAL_REL_IOSTAT.....	2513
13.2.4.8 GLOBAL_REL_IOSTAT.....	2513
13.2.4.9 SUMMARY_REL_IOSTAT.....	2514
13.2.5 Object.....	2514
13.2.5.1 STAT_USER_TABLES.....	2514

13.2.5.2 SUMMARY_STAT_USER_TABLES.....	2516
13.2.5.3 GLOBAL_STAT_USER_TABLES.....	2517
13.2.5.4 STAT_USER_INDEXES.....	2518
13.2.5.5 SUMMARY_STAT_USER_INDEXES.....	2519
13.2.5.6 GLOBAL_STAT_USER_INDEXES.....	2519
13.2.5.7 STAT_SYS_TABLES.....	2520
13.2.5.8 SUMMARY_STAT_SYS_TABLES.....	2521
13.2.5.9 GLOBAL_STAT_SYS_TABLES.....	2523
13.2.5.10 STAT_SYS_INDEXES.....	2524
13.2.5.11 SUMMARY_STAT_SYS_INDEXES.....	2525
13.2.5.12 GLOBAL_STAT_SYS_INDEXES.....	2525
13.2.5.13 STAT_ALL_TABLES.....	2526
13.2.5.14 SUMMARY_STAT_ALL_TABLES.....	2527
13.2.5.15 GLOBAL_STAT_ALL_TABLES.....	2528
13.2.5.16 STAT_ALL_INDEXES.....	2530
13.2.5.17 SUMMARY_STAT_ALL_INDEXES.....	2530
13.2.5.18 GLOBAL_STAT_ALL_INDEXES.....	2531
13.2.5.19 STAT_DATABASE.....	2531
13.2.5.20 SUMMARY_STAT_DATABASE.....	2533
13.2.5.21 GLOBAL_STAT_DATABASE.....	2534
13.2.5.22 STAT_DATABASE_CONFLICTS.....	2536
13.2.5.23 SUMMARY_STAT_DATABASE_CONFLICTS.....	2536
13.2.5.24 GLOBAL_STAT_DATABASE_CONFLICTS.....	2537
13.2.5.25 STAT_XACT_ALL_TABLES.....	2537
13.2.5.26 SUMMARY_STAT_XACT_ALL_TABLES.....	2538
13.2.5.27 GLOBAL_STAT_XACT_ALL_TABLES.....	2538
13.2.5.28 STAT_XACT_SYS_TABLES.....	2539
13.2.5.29 SUMMARY_STAT_XACT_SYS_TABLES.....	2540
13.2.5.30 GLOBAL_STAT_XACT_SYS_TABLES.....	2540
13.2.5.31 STAT_XACT_USER_TABLES.....	2541
13.2.5.32 SUMMARY_STAT_XACT_USER_TABLES.....	2542
13.2.5.33 GLOBAL_STAT_XACT_USER_TABLES.....	2542
13.2.5.34 STAT_XACT_USER_FUNCTIONS.....	2543
13.2.5.35 SUMMARY_STAT_XACT_USER_FUNCTIONS.....	2544
13.2.5.36 GLOBAL_STAT_XACT_USER_FUNCTIONS.....	2544
13.2.5.37 STAT_BAD_BLOCK.....	2545
13.2.5.38 SUMMARY_STAT_BAD_BLOCK.....	2545
13.2.5.39 GLOBAL_STAT_BAD_BLOCK.....	2546
13.2.5.40 STAT_USER_FUNCTIONS.....	2546
13.2.5.41 SUMMARY_STAT_USER_FUNCTIONS.....	2547
13.2.5.42 GLOBAL_STAT_USER_FUNCTIONS.....	2547
13.2.6 Workload.....	2548

13.2.6.1	WORKLOAD_SQL_COUNT.....	2548
13.2.6.2	SUMMARY_WORKLOAD_SQL_COUNT.....	2548
13.2.6.3	WORKLOAD_TRANSACTION.....	2549
13.2.6.4	SUMMARY_WORKLOAD_TRANSACTION.....	2550
13.2.6.5	GLOBAL_WORKLOAD_TRANSACTION.....	2551
13.2.6.6	WORKLOAD_SQL_ELAPSE_TIME.....	2552
13.2.6.7	SUMMARY_WORKLOAD_SQL_ELAPSE_TIME.....	2553
13.2.6.8	USER_TRANSACTION.....	2554
13.2.6.9	GLOBAL_USER_TRANSACTION.....	2555
13.2.7	Session and Thread.....	2555
13.2.7.1	SESSION_STAT.....	2555
13.2.7.2	GLOBAL_SESSION_STAT.....	2556
13.2.7.3	SESSION_TIME.....	2556
13.2.7.4	GLOBAL_SESSION_TIME.....	2557
13.2.7.5	SESSION_MEMORY.....	2557
13.2.7.6	GLOBAL_SESSION_MEMORY.....	2557
13.2.7.7	SESSION_MEMORY_DETAIL.....	2558
13.2.7.8	GLOBAL_SESSION_MEMORY_DETAIL.....	2558
13.2.7.9	SESSION_STAT_ACTIVITY.....	2559
13.2.7.10	GLOBAL_SESSION_STAT_ACTIVITY.....	2562
13.2.7.11	THREAD_WAIT_STATUS.....	2565
13.2.7.12	GLOBAL_THREAD_WAIT_STATUS.....	2566
13.2.7.13	LOCAL_THREADPOOL_STATUS.....	2567
13.2.7.14	GLOBAL_THREADPOOL_STATUS.....	2568
13.2.7.15	SESSION_CPU_RUNTIME.....	2568
13.2.7.16	SESSION_MEMORY_RUNTIME.....	2569
13.2.7.17	LOCAL_ACTIVE_SESSION.....	2570
13.2.7.18	GLOBAL_ACTIVE_SESSION.....	2572
13.2.8	Transaction.....	2574
13.2.8.1	TRANSACTIONS_RUNNING_XACTS.....	2574
13.2.8.2	SUMMARY_TRANSACTIONS_RUNNING_XACTS.....	2574
13.2.8.3	GLOBAL_TRANSACTIONS_RUNNING_XACTS.....	2575
13.2.8.4	TRANSACTIONS_PREPARED_XACTS.....	2575
13.2.8.5	SUMMARY_TRANSACTIONS_PREPARED_XACTS.....	2576
13.2.8.6	GLOBAL_TRANSACTIONS_PREPARED_XACTS.....	2576
13.2.9	Query.....	2577
13.2.9.1	STATEMENT.....	2577
13.2.9.2	SUMMARY_STATEMENT.....	2581
13.2.9.3	STATEMENT_COUNT.....	2584
13.2.9.4	GLOBAL_STATEMENT_COUNT.....	2585
13.2.9.5	SUMMARY_STATEMENT_COUNT.....	2587
13.2.9.6	STATEMENT_HISTORY.....	2588

13.2.10 Cache and I/O.....	2593
13.2.10.1 STATIO_USER_TABLES.....	2593
13.2.10.2 SUMMARY_STATIO_USER_TABLES.....	2594
13.2.10.3 GLOBAL_STATIO_USER_TABLES.....	2594
13.2.10.4 STATIO_USER_INDEXES.....	2595
13.2.10.5 SUMMARY_STATIO_USER_INDEXES.....	2596
13.2.10.6 GLOBAL_STATIO_USER_INDEXES.....	2596
13.2.10.7 STATIO_USER_SEQUENCES.....	2597
13.2.10.8 SUMMARY_STATIO_USER_SEQUENCES.....	2597
13.2.10.9 GLOBAL_STATIO_USER_SEQUENCES.....	2597
13.2.10.10 STATIO_SYS_TABLES.....	2598
13.2.10.11 SUMMARY_STATIO_SYS_TABLES.....	2599
13.2.10.12 GLOBAL_STATIO_SYS_TABLES.....	2599
13.2.10.13 STATIO_SYS_INDEXES.....	2600
13.2.10.14 SUMMARY_STATIO_SYS_INDEXES.....	2600
13.2.10.15 GLOBAL_STATIO_SYS_INDEXES.....	2601
13.2.10.16 STATIO_SYS_SEQUENCES.....	2601
13.2.10.17 SUMMARY_STATIO_SYS_SEQUENCES.....	2602
13.2.10.18 GLOBAL_STATIO_SYS_SEQUENCES.....	2602
13.2.10.19 STATIO_ALL_TABLES.....	2603
13.2.10.20 SUMMARY_STATIO_ALL_TABLES.....	2603
13.2.10.21 GLOBAL_STATIO_ALL_TABLES.....	2604
13.2.10.22 STATIO_ALL_INDEXES.....	2605
13.2.10.23 SUMMARY_STATIO_ALL_INDEXES.....	2605
13.2.10.24 GLOBAL_STATIO_ALL_INDEXES.....	2606
13.2.10.25 STATIO_ALL_SEQUENCES.....	2606
13.2.10.26 SUMMARY_STATIO_ALL_SEQUENCES.....	2607
13.2.10.27 GLOBAL_STATIO_ALL_SEQUENCES.....	2607
13.2.11 Communication Library.....	2607
13.2.11.1 COMM_DELAY.....	2607
13.2.11.2 GLOBAL_COMM_DELAY.....	2608
13.2.11.3 COMM_RECV_STREAM.....	2609
13.2.11.4 GLOBAL_COMM_RECV_STREAM.....	2609
13.2.11.5 COMM_SEND_STREAM.....	2610
13.2.11.6 GLOBAL_COMM_SEND_STREAM.....	2611
13.2.11.7 COMM_STATUS.....	2612
13.2.11.8 GLOBAL_COMM_STATUS.....	2613
13.2.12 Utility.....	2614
13.2.12.1 REPLICATION_STAT.....	2614
13.2.12.2 GLOBAL_REPLICATION_STAT.....	2615
13.2.12.3 REPLICATION_SLOTS.....	2616
13.2.12.4 GLOBAL_REPLICATION_SLOTS.....	2617

13.2.12.5 PARALLEL_DECODE_STATUS.....	2618
13.2.12.6 GLOBAL_PARALLEL_DECODE_STATUS.....	2618
13.2.12.7 PARALLEL_DECODE_THREAD_INFO.....	2619
13.2.12.8 GLOBAL_PARALLEL_DECODE_THREAD_INFO.....	2620
13.2.12.9 BGWRITER_STAT.....	2620
13.2.12.10 GLOBAL_BGWRITER_STAT.....	2621
13.2.12.11 POOLER_STATUS.....	2622
13.2.12.12 GLOBAL_COMM_CHECK_CONNECTION_STATUS.....	2623
13.2.12.13 GLOBAL_CKPT_STATUS.....	2624
13.2.12.14 GLOBAL_DOUBLE_WRITE_STATUS.....	2624
13.2.12.15 GLOBAL_PAGEWRITER_STATUS.....	2625
13.2.12.16 GLOBAL_POOLER_STATUS.....	2626
13.2.12.17 GLOBAL_RECORD_RESET_TIME.....	2626
13.2.12.18 GLOBAL_REDO_STATUS.....	2627
13.2.12.19 GLOBAL_RECOVERY_STATUS.....	2628
13.2.12.20 CLASS_VITAL_INFO.....	2629
13.2.12.21 USER_LOGIN.....	2629
13.2.12.22 SUMMARY_USER_LOGIN.....	2630
13.2.12.23 GLOBAL_GET_BGWRITER_STATUS.....	2630
13.2.12.24 GLOBAL_SINGLE_FLUSH_DW_STATUS.....	2630
13.2.12.25 GLOBAL_CANDIDATE_STATUS.....	2631
13.2.13 Lock.....	2632
13.2.13.1 LOCKS.....	2632
13.2.13.2 GLOBAL_LOCKS.....	2633
13.2.14 Wait Event.....	2635
13.2.14.1 WAIT_EVENTS.....	2635
13.2.14.2 GLOBAL_WAIT_EVENTS.....	2636
13.2.14.3 WAIT_EVENT_INFO.....	2636
13.2.15 Configuration.....	2650
13.2.15.1 CONFIG_SETTINGS.....	2650
13.2.15.2 GLOBAL_CONFIG_SETTINGS.....	2651
13.2.16 Operator.....	2652
13.2.16.1 OPERATOR_HISTORY_TABLE.....	2653
13.2.16.2 OPERATOR_HISTORY.....	2654
13.2.16.3 OPERATOR_RUNTIME.....	2654
13.2.16.4 GLOBAL_OPERATOR_HISTORY.....	2656
13.2.16.5 GLOBAL_OPERATOR_HISTORY_TABLE.....	2657
13.2.16.6 GLOBAL_OPERATOR_RUNTIME.....	2657
13.2.17 Workload Manager.....	2659
13.2.17.1 WLM_CGROUP_CONFIG.....	2659
13.2.17.2 WLM_CLUSTER_RESOURCE_RUNTIME.....	2659
13.2.17.3 WLM_CONTROLGROUP_CONFIG.....	2660

13.2.17.4 WLM_RESOURCEPOOL_RUNTIME.....	2660
13.2.17.5 WLM_USER_RESOURCE_CONFIG.....	2661
13.2.17.6 WLM_USER_RESOURCE_RUNTIME.....	2662
13.2.17.7 WLM_WORKLOAD_RUNTIME.....	2663
13.2.17.8 GLOBAL_WLM_WORKLOAD_RUNTIME.....	2664
13.2.17.9 LOCAL_IO_WAIT_INFO.....	2664
13.2.17.10 GLOBAL_IO_WAIT_INFO.....	2665
13.2.18 Global Plan Cache.....	2666
13.2.18.1 LOCAL_PLANCACHE_STATUS.....	2666
13.2.18.2 GLOBAL_PLANCACHE_STATUS.....	2666
13.2.19 RTO & RPO.....	2667
13.2.19.1 global_rto_status.....	2667
13.2.19.2 global_streaming_hadr_rto_and_rpo_stat.....	2667
13.2.20 AI Watchdog.....	2668
13.2.20.1 ai_watchdog_monitor_status.....	2669
13.2.20.2 ai_watchdog_detection_warnings.....	2670
13.2.20.3 ai_watchdog_parameters.....	2671
13.2.21 Discarded.....	2671
13.2.21.1 Query.....	2671
13.2.21.1.1 GS_SLOW_QUERY_INFO.....	2671
13.2.21.1.2 GS_SLOW_QUERY_HISTORY.....	2673
13.2.21.1.3 GLOBAL_SLOW_QUERY_HISTORY.....	2673
13.2.21.1.4 GLOBAL_SLOW_QUERY_INFO.....	2673
13.2.21.2 Global Plancache.....	2673
13.2.21.2.1 LOCAL_PREPARE_STATEMENT_STATUS.....	2673
13.2.21.2.2 GLOBAL_PREPARE_STATEMENT_STATUS.....	2674
13.3 DBE_SQL_UTIL Schema.....	2674
13.3.1 DBE_SQL_UTIL.create_hint_sql_patch.....	2674
13.3.2 DBE_SQL_UTIL.create_abort_sql_patch.....	2675
13.3.3 DBE_SQL_UTIL.drop_sql_patch.....	2676
13.3.4 DBE_SQL_UTIL.enable_sql_patch.....	2676
13.3.5 DBE_SQL_UTIL.disable_sql_patch.....	2676
13.3.6 DBE_SQL_UTIL.show_sql_patch.....	2677
13.3.7 DBE_SQL_UTIL.create_hint_sql_patch.....	2677
13.3.8 DBE_SQL_UTIL.create_abort_sql_patch.....	2678
13.3.9 DBE_SQL_UTIL.create_remote_hint_sql_patch.....	2679
13.3.10 DBE_SQL_UTIL.create_remote_abort_sql_patch.....	2679
13.3.11 DBE_SQL_UTIL.drop_remote_sql_patch.....	2680
13.3.12 DBE_SQL_UTIL.enable_remote_sql_patch.....	2680
13.3.13 DBE_SQL_UTIL.disable_remote_sql_patch.....	2681
14 Configuring Running Parameters.....	2682
14.1 Viewing Parameters.....	2682

14.2 Setting Parameters.....	2683
14.3 GUC Parameters.....	2685
14.3.1 GUC Parameter Usage.....	2685
14.3.2 File Location.....	2686
14.3.3 Connection and Authentication.....	2688
14.3.3.1 Connection Settings.....	2688
14.3.3.2 Security and Authentication (postgresql.conf).....	2696
14.3.3.3 Communication Library Parameters.....	2707
14.3.4 Resource Consumption.....	2718
14.3.4.1 Memory.....	2718
14.3.4.2 Disk Space.....	2730
14.3.4.3 Kernel Resource Usage.....	2731
14.3.4.4 Cost-based Vacuum Delay.....	2732
14.3.4.5 Background Writer.....	2733
14.3.4.6 Asynchronous I/O.....	2737
14.3.5 Parallel Data Import.....	2738
14.3.6 Write Ahead Log.....	2740
14.3.6.1 Settings.....	2740
14.3.6.2 Checkpoints.....	2748
14.3.6.3 Log Replay.....	2751
14.3.6.4 Archiving.....	2754
14.3.7 HA Replication.....	2757
14.3.7.1 Sending Server.....	2757
14.3.7.2 Primary Server.....	2765
14.3.7.3 Standby Server.....	2772
14.3.8 Query Planning.....	2777
14.3.8.1 Optimizer Method Configuration.....	2777
14.3.8.2 Optimizer Cost Constants.....	2789
14.3.8.3 Genetic Query Optimizer.....	2792
14.3.8.4 Other Optimizer Options.....	2795
14.3.9 Error Reporting and Logging.....	2818
14.3.9.1 Logging Destination.....	2818
14.3.9.2 Logging Time.....	2822
14.3.9.3 Logging Content.....	2825
14.3.9.4 Using CSV Log Output.....	2835
14.3.10 Alarm Detection.....	2837
14.3.11 Statistics During the Database Running.....	2839
14.3.11.1 Query and Index Statistics Collector.....	2839
14.3.11.2 Performance Statistics.....	2842
14.3.11.3 Hotspot Key Statistics.....	2843
14.3.12 Automatic Vacuuming.....	2844
14.3.13 Default Settings of Client Connection.....	2848

14.3.13.1 Statement Behavior.....	2848
14.3.13.2 Locale and Formatting.....	2855
14.3.13.3 Other Default Parameters.....	2860
14.3.14 Lock Management.....	2862
14.3.15 Version and Platform Compatibility.....	2866
14.3.15.1 Compatibility with Earlier Versions.....	2866
14.3.15.2 Platform and Client Compatibility.....	2870
14.3.16 Fault Tolerance.....	2889
14.3.17 Connection Pool Parameters.....	2891
14.3.18 Cluster Transaction Parameters.....	2896
14.3.19 Dual-Cluster Replication Parameters.....	2905
14.3.20 Developer Options.....	2908
14.3.21 Auditing.....	2919
14.3.21.1 Audit Switch.....	2919
14.3.21.2 User and Permission Audit.....	2923
14.3.21.3 Operation Auditing.....	2925
14.3.22 Transaction Monitoring.....	2934
14.3.23 CM Parameters.....	2935
14.3.23.1 CM Agent Parameters.....	2935
14.3.23.2 CM Server Parameters.....	2942
14.3.24 GTM Parameters.....	2954
14.3.25 Upgrade Parameters.....	2961
14.3.26 Miscellaneous Parameters.....	2962
14.3.27 Wait Event.....	2968
14.3.28 Query.....	2968
14.3.29 System Performance Snapshot.....	2974
14.3.30 Security Configuration.....	2977
14.3.31 HyperLogLog.....	2978
14.3.32 User-defined Functions.....	2981
14.3.33 Collaborative Analysis.....	2981
14.3.34 Acceleration Cluster.....	2982
14.3.35 Scheduled Task.....	2983
14.3.36 Thread Pool.....	2984
14.3.37 Full Text Search.....	2987
14.3.38 Backup and Restoration.....	2988
14.3.39 AI Features.....	2989
14.3.40 Global SysCache Parameters.....	2994
14.3.41 Reserved Parameters.....	2994
14.3.42 Read Parameters of the Standby Node in a Distributed System.....	2995
14.3.43 Restoring Data on the Standby Node.....	2996
14.3.44 Undo.....	2996
14.3.45 Rollback Parameters.....	2997

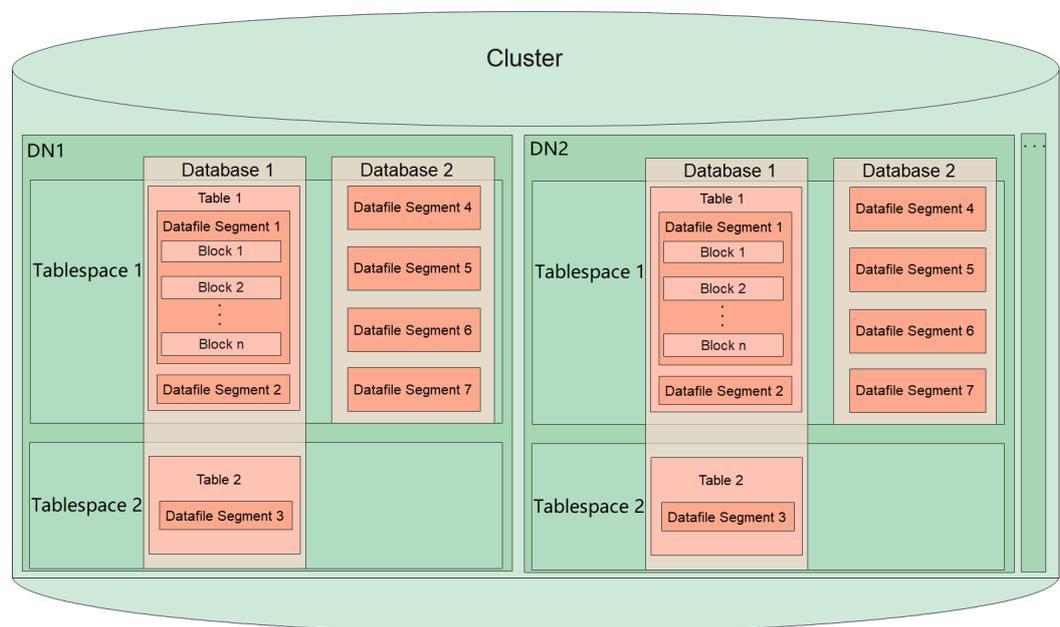
14.3.46 DCF Parameters Settings.....	2997
14.3.47 Flashback.....	3006

1 Database System Overview

1.1 Database Logical Architecture

Each DN in a cluster stores data on disks. This section describes the objects on each DN from the logical view, the relationship between these objects, and how data is distributed on different nodes. [Figure 1-1](#) shows the database logical structure.

Figure 1-1 Database Logical Architecture

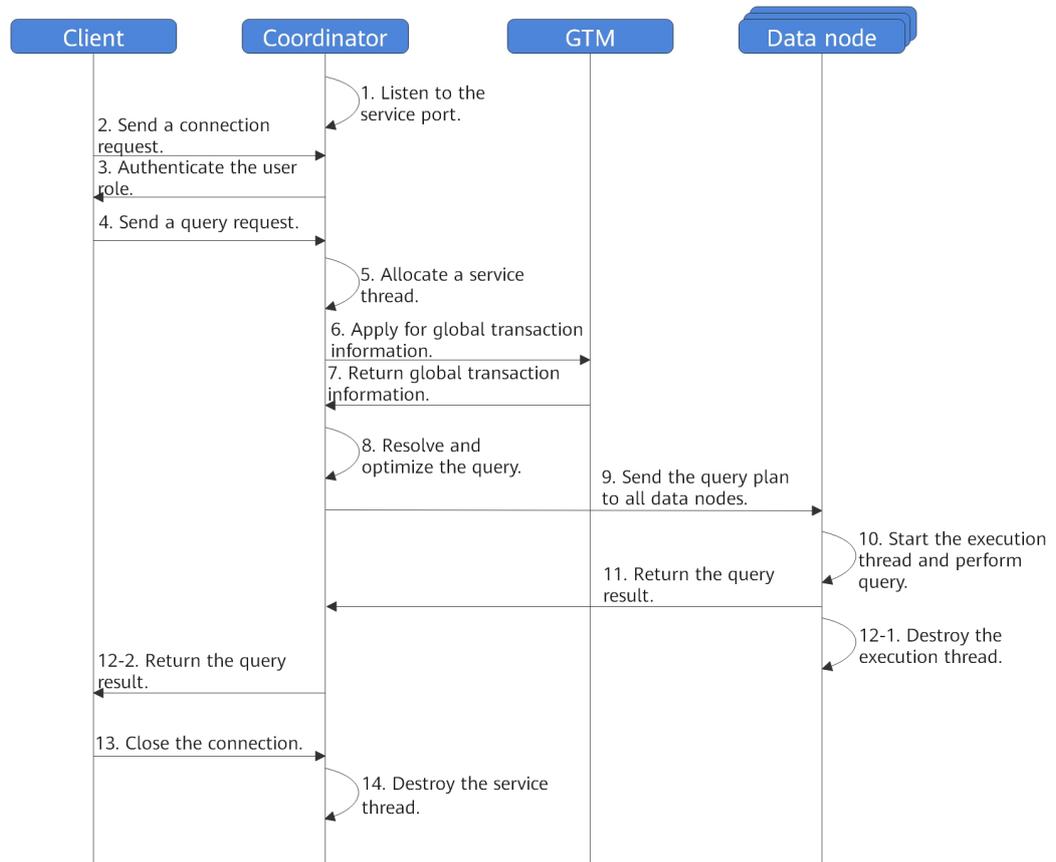


NOTE

- A tablespace is a directory. A cluster contains one or more tablespaces to store physical files of the database. Each tablespace can contain files belonging to different databases.
- A database manages various data objects and is isolated from each other. Objects managed by a database can be distributed to multiple tablespaces.
- A datafile segment stores data of only one table. A table containing more than 1 GB of data is stored in multiple datafile segments.
- One table belongs to only one database and one tablespace. The datafile segments storing the data of the same table must be in the same tablespace.
- A block is a basic unit for database management. Its default size is 8 KB.
- Data can be distributed on DN in replication, hash, range, or list mode. You can specify a mode when creating a table.

1.2 Query Request Handling Process

Figure 1-2 GaussDB service response process



1.3 Managing Transactions

A transaction is a customized sequence of database operations, which form an integral unit of work. In GaussDB, you can start, set, commit, and roll back transactions. GaussDB supports the following transaction isolation levels: READ COMMITTED, REPEATABLE READ, and SERIALIZABLE. SERIALIZABLE is equivalent to REPEATABLE READ.

Controlling Transactions

Transaction operations supported by a database are as follows:

- Starting transactions
You can use the `START TRANSACTION` or `BEGIN` syntax to start a transaction. For details, see [START TRANSACTION](#) and [BEGIN](#).
- Setting transactions
You can use the `SET TRANSACTION` or `SET LOCAL TRANSACTION` syntax to set transactions. For details, see [SET TRANSACTION](#).
- Committing transactions
You can commit all operations of a transaction using `COMMIT` or `END`. For details, see [COMMIT | END](#).
- Rolling back transactions
Rollback indicates that the system cancels all changes that a transaction has made to a database if the transaction fails to be executed due to a fault. For details, see [ROLLBACK](#).

Transaction Isolation Levels

A transaction isolation level specifies how concurrent transactions process the same object.

NOTE

The isolation level cannot be changed after the first data manipulation statement (`SELECT`, `INSERT`, `DELETE`, `UPDATE`, `FETCH`, or `COPY`) in a transaction is executed.

- **READ COMMITTED:** At this level, a transaction can access only committed data. This is the default level.
The `SELECT` statement accesses the snapshot of the database taken when the query begins. The `SELECT` statement can also access the data modifications in its transaction, regardless of whether they have been committed. Note that different database snapshots may be available to two consecutive `SELECT` statements for the same transaction, because data may be committed for other transactions while the first `SELECT` statement is executed.
At the **READ COMMITTED** level, the execution of each statement begins with a new snapshot, which contains all the transactions that have been committed by the execution time. Therefore, during a transaction, a statement can access the result of other committed transactions. Check whether a single statement always accesses absolutely consistent data in a database.
Transaction isolation at this level meets the requirements of many applications, and is fast and easy to use. However, applications performing complicated queries and updates may require data that is more consistent than this level can provide.
- **REPEATABLE READ:** At this level, a transaction can only read data committed before it starts. Uncommitted data or data committed in other concurrent transactions cannot be read. However, a query can read earlier data modifications in its transaction, regardless of whether they have been committed. **READ COMMITTED** differs from this level in that a transaction reads the snapshot taken at the start of the transaction, not at the beginning of the current query within the transaction. Therefore, the `SELECT` statement

within a transaction always reads the same data, and cannot read data committed by other concurrent transactions after the transaction starts. Applications at this level must be able to retry transactions, because serialization failures may occur.

- **SERIALIZABLE:** Currently, GaussDB does not support this isolation level. Setting this isolation level is equivalent to REPEATABLE READ.

 NOTE

REPEATABLE READ is implemented based on multi-version snapshots and write skew may occur. To avoid this scenario, perform the SELECT FOR UPDATE operation on the rows involved in the transaction. An example of write skew is as follows:

Scenario 1: Table **a** has the **id** and **value** columns of the int type. Two data records are inserted. Assume that the sum of the values of the two data records must be less than or equal to 10 in the service logic of table **a**. Two transactions are concurrently started. The values are updated and modified based on the read values. After the modification, the sum of values is less than or equal to 10 in the transactions. After the modification is committed, the sum of values is 12, which violates the assumed service logic of table **a**.

```
gaussdb=# create table a(id int, value int);
CREATE TABLE
gaussdb=# insert into a values(1,4);
INSERT 0 1
gaussdb=# insert into a values(2,4);
INSERT 0 1
session1 :
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-----
 1 |  4
 2 |  4
(2 rows)
gaussdb=# update a set value = 6 where id = 1;
UPDATE 1
gaussdb=# select * from a;
id | value
----+-----
 1 |  6
 2 |  4
(2 rows)
session2:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-----
 1 |  4
 2 |  4
(2 rows)
gaussdb=# update a set value = 6 where id = 2;
UPDATE 1
gaussdb=# select * from a;
id | value
----+-----
 1 |  4
 2 |  6
(2 rows)
session1:
gaussdb=# commit;
COMMIT
session2:
gaussdb=# commit;
COMMIT
gaussdb=# select * from a;
id | value
----+-----
 1 |  6
 2 |  6
(2 rows)
```

Scenario 2: Table **a** has the **id** and **value** columns of the int type. The **id** is the primary key. When the primary key is deleted and inserted concurrently, the values of two primary keys may be read in the transaction, violating the primary key constraint.

```
gaussdb=# create table a(id int primary key, value int);
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "a_pkey" for table "a"
CREATE TABLE
gaussdb=# insert into a values(1,10);
INSERT 0 1
session1:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# delete a where id = 1;
DELETE 1
session2:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-----
1 | 10
(1 row)
session1:
gaussdb=# commit;
COMMIT
session2:
gaussdb=# insert into a values(1, 100);
INSERT 0 1
gaussdb=# select * from a;
id | value
----+-----
1 | 10
1 | 100
(2 rows)
```

1.4 Concepts

Databases

Databases manage various data objects and are isolated from each other. While creating a database, you can specify a tablespace. If you do not specify it, the object will be saved to the PG_DEFAULT tablespace by default. Objects managed by a database can be distributed to multiple tablespaces.

Tablespaces

In GaussDB, a tablespace is a directory storing physical files of the databases. Multiple tablespaces can coexist in a cluster. Files are physically isolated using tablespaces and managed by a file system.

Schemas

GaussDB schemas logically separate databases. All database objects are created under certain schemas. In GaussDB, schemas and users are loosely bound. When you create a user, a schema with the same name as the user will be created automatically. You can also create a schema or specify another schema.

Users and Roles

GaussDB uses users and roles to control the access to databases. A role can be a database user or a group of database users, depending on role settings. Each user

can have only one role. In GaussDB, the difference between roles and users is that a role does not have the LOGIN permission by default. In GaussDB, you can put a user's role under a parent role for flexible management.

Transactions

In GaussDB, transactions are managed by multi-version concurrency control (MVCC) and two-phase locking (2PL). It enables smooth data reads and writes. In GaussDB, an Astore stores historical version data together with the current tuple version. The GaussDB Astore uses a VACUUM thread instead of rollback segments to periodically delete historical version data. Generally, you do not need to pay special attention to the VACUUM thread unless you need to optimize the performance. The GaussDB Ustore stores historical version data to the undo rollback segments. The thread for purging undo deletes historical version data in a unified manner. In addition, GaussDB automatically commits transactions for single-statement queries (without using statements such as BEGIN to explicitly start a transaction block).

TOAST

The Oversized-Attribute Storage Technique (TOAST) is a special storage technique used to overcome the limitation that a single field in the database is too large to be directly stored in a standard data page. GaussDB uses a fixed page size, typically 8 KB. When the size of a row of data exceeds the page size, the database employs TOAST to ensure that these oversized fields can be effectively stored and managed. Out-of-line storage is a part of TOAST. This technology compresses or slices large field values into multiple physical rows and stores them in another table (known as the TOAST table). The main table retains only references (usually pointers) to these large field values. If a table has an associated TOAST table, the **reltoastrelid** column in the PG_CLASS system catalog records the OID of the TOAST table. If the table has no such an associated TOAST table, the value of **reltoastrelid** is 0.

2 Database Security

2.1 Users and Permissions

2.1.1 Default Permission Mechanism

A user who creates an object is the owner of this object. By default, [separation of duties](#) is disabled after cluster installation. A database system administrator has the same permissions as object owners. After an object is created, only the object owner or system administrators can query, modify, and delete the object, and grant permissions for the object to other users through [GRANT](#) by default.

To enable another user to use the object, grant required permissions to the user or the role that contains the user.

GaussDB supports the following permissions: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, CREATE, CONNECT, EXECUTE, USAGE, ALTER, DROP, COMMENT, INDEX, and VACUUM. Permission types are associated with object types. For permission details, see [GRANT](#).

To revoke permissions, use [REVOKE](#). Object owners have implicit permissions (such as ALTER, DROP, COMMENT, INDEX, VACUUM, GRANT, and REVOKE) on objects. That is, once becoming the owner of an object, the owner is immediately granted the implicit permissions on the object. Object owners can remove their own common permissions (SELECT, INSERT, UPDATE, and DELETE), for example, making tables read-only to themselves or others, except system administrators.

System catalogs and views are visible to either system administrators or all users. System catalogs and views that require system administrator permissions can be queried only by system administrators. For details, see [System Catalogs and System Views](#).

The database provides the object isolation feature. If this feature is enabled, users can view only the objects (tables, views, columns, and functions) that they have the permission to access. System administrators are not affected by this feature. For details, see [ALTER DATABASE](#).

It is not recommended to modify the permissions on system catalogs and system views.

2.1.2 Administrators

Initial User

The account automatically generated during the cluster installation is called the initial user. The initial user is also a system administrator, security administrator, audit administrator, monitor administrator, O&M administrator, and security policy administrator. It has the highest permissions in the system and can perform all operations. If the initial username is not set during the installation, the username is the same as the name of the OS user who installs the cluster. If the password of the initial user is not set during the cluster installation, the password is empty after the installation. In this case, you need to set the password of the initial user on the gsql client before performing other operations. If the initial user password is empty, you cannot perform other SQL operations, such as upgrade, capacity expansion, and node replacement, except changing the password.

An initial user bypasses all permission checks. You are advised to use an initial user as a database administrator only for database management other than service running.

System Administrator

A system administrator is an account with the **SYSADMIN** attribute. By default, a system administrator has the same permissions as the object owner but does not have the object permissions in the db_perf schema or the permission to use Roach to perform backup and restoration.

To create a database administrator, connect to the database as the initial user or a system administrator and use the **CREATE USER** or **ALTER USER** statement with the **SYSADMIN** option.

```
gaussdb=# CREATE USER sysadmin WITH SYSADMIN password '*****';
```

Or

```
gaussdb=# ALTER USER joe SYSADMIN;
```

When ALTER USER is executed, the user must exist.

Security Administrator

A security administrator is an account with the **CREATEROLE** attribute, which has the permission to create, modify, and delete users or roles.

If you want to create a security administrator and separation of duties is disabled, connect to the database as a system administrator or security administrator. If separation of duties is enabled, connect to the database as a security administrator and use the **CREATE USER** or **ALTER USER** statement with the **CREATEROLE** option.

```
gaussdb=# CREATE USER createrole WITH CREATEROLE password '*****';
```

Or

```
gaussdb=# ALTER USER joe CREATEROLE;
```

When ALTER USER is executed, the user must exist.

Audit Administrator

An audit administrator is an account with the **AUDITADMIN** attribute, which has the permission to view and delete audit logs.

If you want to create an audit administrator and separation of duties is disabled, connect to the database as a system administrator or security administrator. If separation of duties is enabled, connect to the database only as the initial user and use the **CREATE USER** or **ALTER USER** statement with the **AUDITADMIN** option.

```
gaussdb=# CREATE USER auditadmin WITH AUDITADMIN password "*****";
```

Or

```
gaussdb=# ALTER USER joe AUDITADMIN;
```

When ALTER USER is executed, the user must exist.

Monitor Administrator

A monitor administrator is an account with the **MONADMIN** attribute and has the permission to view views and functions in the `dbe_perf` schema. The monitor administrator can also grant or revoke object permissions in the `dbe_perf` schema.

To create a monitor administrator, connect to the database as a system administrator and use the **CREATE USER** or **ALTER USER** statement with the **MONADMIN** option.

```
gaussdb=# CREATE USER monadmin WITH MONADMIN password "*****";
```

Or

```
gaussdb=# ALTER USER joe MONADMIN;
```

When ALTER USER is executed, the user must exist.

O&M Administrator

An O&M administrator is an account with the **OPRADMIN** attribute and has the permission to use Roach to perform backup and restoration.

To create an O&M administrator, connect to the database as an initial user and use the **CREATE USER** or **ALTER USER** statement with the **OPRADMIN** option.

```
gaussdb=# CREATE USER opradmin WITH OPRADMIN password "*****";
```

Or

```
gaussdb=# ALTER USER joe OPRADMIN;
```

When ALTER USER is executed, the user must exist.

Security Policy Administrator

A security policy administrator is an account with the **POLADMIN** attribute and has the permission to create resource tags, masking policies, and unified audit policies.

To create a security policy administrator, connect to the database as an administrator and use the **CREATE USER** or **ALTER USER** statement with the **POLADMIN** option.

```
gaussdb=# CREATE USER poladmin WITH POLADMIN password "*****";
```

Or

```
gaussdb=# ALTER USER joe POLADMIN;
```

When ALTER USER is executed, the user must exist.

2.1.3 Separation of Duties

Descriptions in **Default Permission Mechanism** and **Administrators** are about the initial situation after a cluster is created. By default, the system administrator with the **SYSADMIN** attribute has the highest permission in the system.

To avoid risks caused by centralized permissions, you can enable separation of duties to assign the system administrator's user management permission to security administrators and audit management permission to audit administrators.

After separation of duties, the system administrator does not have the **CREATEROLE** attribute (security administrator) or the **AUDITADMIN** attribute (audit administrator). That is, the system administrator can neither create roles or users, nor view or maintain database audit logs. For details about the **CREATEROLE** and **AUDITADMIN** attributes, see **CREATE ROLE**.

Separation of duties does not take effect for an initial user. Therefore, you are advised to use an initial user as a database administrator only for database management other than service running.

Contact Huawei technical support.

WARNING

If you need to use the separation of duties model, specify it during database initialization. You are advised not to switch the permission management model back and forth. In particular, if you want to switch from a non-separation-of-duties permission management model to the separation-of-duties permission management model, you need to review the permission set of existing users. If a user has the system administrator permission and audit administrator permission, the permissions need to be tailored.

After separation of duties, the system administrator does not have permissions for non-system schemas of other users. Therefore, the system administrator cannot access the objects in other users' schemas before being granted the permissions. For details about permission changes before and after enabling separation of duties, see **Table 2-1** and **Table 2-2**.

Table 2-1 Default user permissions

Object Name	Initial User (ID: 10)	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespaces	Has all permissions.	Can create, modify, delete, access, or grant permissions for tablespaces.	Cannot create, modify, delete, or grant permissions for tablespaces and can access tablespaces if the access permission is granted.		
Schemas		Has all permissions for all schemas except db_perf.	Has all permissions for their own schemas, but does not have permissions for non-system schemas of other users.		
User-defined functions		Has all permissions for all user-defined functions.	Has all permissions for their own functions, and has only the call permission for other users' functions.		
User-defined tables or views		Has all permissions for all user-defined tables or views.	Has all permissions for their own tables or views, but does not have permissions for other users' tables or views.		

Table 2-2 Changes in permissions after separation of duties

Object Name	Initial User (ID: 10)	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespaces	No change. Has all permissions.	No change.	No change.		
Schemas		Permissions reduced Has all permissions for their own schemas, but does not have permissions for non-system schemas of other users.	No change.		
User-defined functions		Cannot access functions in non-system schemas of other users before being granted the permissions.	No change.		

Object Name	Initial User (ID: 10)	System Administrator	Security Administrator	Audit Administrator	Common User
User-defined tables or views		Cannot access tables or views in non-system schemas of other users before being granted the permissions.	No change.		

NOTICE

PG_STATISTIC and PG_STATISTIC_EXT store sensitive information about statistical objects, such as high-frequency MCVs. After separation of duties is enabled, the system administrator can still access the two system catalogs to obtain sensitive information in the statistics.

2.1.4 Users

You can use CREATE USER and ALTER USER to create and manage database users, respectively. A database cluster can have one or more databases. Users and roles are shared within the entire cluster, but their data is not shared. That is, a user can connect to any database, but after the connection is successful, any user can access only the database declared in the connection request.

In modes other than **separation of duties**, GaussDB user accounts can be created and deleted only by a system administrator or a security administrator with the **CREATEROLE** attribute. In separation-of-duties mode, a user account can be created only by an initial user or a security administrator.

When a user logs in, GaussDB authenticates the user. A user can own databases and database objects (such as tables), and grant permissions of these objects to other users and roles. In addition to system administrators, users with the **CREATEDB** attribute can create databases and grant permissions on these databases.

Adding, Modifying, and Deleting Users

- To create a user, use the SQL statement **CREATE USER**.
For example, create user **joe** and set the **CREATEDB** attribute for the user.
gaussdb=# **CREATE USER joe WITH CREATEDB PASSWORD '*****';**
CREATE ROLE
- To create a system administrator, use the **CREATE USER** statement with the **SYSADMIN** option.
- To delete an existing user, use **DROP USER**.
- To change a user account (for example, rename the user or change the password), use **ALTER USER**.

- To view a user list, query the **PG_USER** view.
`gaussdb=# SELECT * FROM pg_user;`
- To view user attributes, query the **PG_AUTHID** system catalog.
`gaussdb=# SELECT * FROM pg_authid;`

Permanent User

GaussDB provides a permanent user solution. You can create a permanent user with the **PERSISTENCE** attribute, which can use the `service_reserved_connections` channel to connect to the database.

NOTE

service_reserved_connections indicates the minimum number of connections reserved with the **PERSISTENCE** attribute. You are advised not to set this parameter to a large value.

```
gaussdb=# CREATE USER user_persistence WITH PERSISTENCE IDENTIFIED BY "*****";
```

Only the initial user is allowed to create, modify, and delete permanent users with the **PERSISTENCE** attribute.

2.1.5 Roles

After a role is granted to a user through **GRANT**, the user will have all the permissions of the role. It is recommended that roles be used to efficiently grant permissions. For example, you can create different roles of design, development, and maintenance personnel, grant the roles to users, and then grant specific data permissions required by different users. When permissions are granted or revoked at the role level, these changes take effect on all members of the role.

GaussDB provides an implicitly defined group **PUBLIC** that contains all roles. By default, all new users and roles have the permissions of **PUBLIC**. For details about the default permissions of **PUBLIC**, see **GRANT**. To revoke permissions of **PUBLIC** from a user or role, or re-grant these permissions to them, add the **PUBLIC** keyword in the **REVOKE** or **GRANT** statement.

To view all roles, query the system catalog **PG_ROLES**.

```
SELECT * FROM PG_ROLES;
```

Adding, Modifying, and Deleting Roles

In scenarios other than separation of duties, a role can be created, modified, and deleted only by a system administrator or a user with the **CREATEROLE** attribute. In **separation-of-duties** scenarios, a role can be created, modified, and deleted only by an initial user or a user with the **CREATEROLE** attribute.

- To create a role, use **CREATE ROLE**.
- To add or delete users in an existing role, use **ALTER ROLE**.
- To drop a role, use **DROP ROLE**. **DROP ROLE** drops only a role, rather than member users in the role.

Built-in Roles

GaussDB provides a group of default roles whose names start with **gs_role_**. These roles are provided to access to specific, typically high-privileged operations. You

can grant these roles to other users or roles within the database so that they can use specific functions. These roles should be given with great care to ensure that they are used where they are needed. [Table 2-3](#) describes the permissions of built-in roles.

Table 2-3 Built-in roles and their permissions

Roles	Permission
gs_role_signal_backend	Permission to call the pg_cancel_backend, pg_terminate_backend, and pg_terminate_session functions to cancel or terminate other sessions, or to call the pg_terminate_active_session_socket function to close active sessions and the socket connection of the client. However, this role cannot perform operations on sessions of the initial user or PERSISTENCE user.
gs_role_tablespace	Permission to create a tablespace.
gs_role_replication	Permission to call logical replication functions, such as kill_snapshot, pg_create_logical_replication_slot, pg_create_physical_replication_slot, pg_drop_replication_slot, pg_replication_slot_advance, pg_create_physical_replication_slot_extern, pg_logical_slot_get_changes, pg_logical_slot_peek_changes, pg_logical_slot_get_binary_changes, and pg_logical_slot_peek_binary_changes.
gs_role_account_lock	Permission to lock and unlock users. However, this role cannot lock or unlock the initial user or users with the PERSISTENCE attribute.
gs_role_pldebugger	Permission to debug functions in db_epldebugger.
gs_role_public_dblink_drop	Permission to delete public database links.
gs_role_public_dblink_alter	Permission to modify public database links.

The restrictions on built-in roles are as follows:

- The role names starting with **gs_role_** are reserved for built-in roles in the database. Do not create users, roles, or schemas starting with **gs_role_** or rename existing users, roles, or schemas to names starting with **gs_role_**.
- Do not perform ALTER or DROP operations on built-in roles.
- By default, built-in roles do not have the LOGIN permission and do not have preset passwords.
- The gs sql meta-commands **\du** and **\dg** do not display information about built-in roles. However, if **pattern** is set to a specific built-in role, the information is displayed.

- When separation of duties is disabled, the initial user, users with the SYSADMIN permission, and users with the ADMIN OPTION built-in role permission have the permission to perform GRANT and REVOKE operations on built-in roles. When separation-of-duties is enabled, the initial user and users with the ADMIN OPTION built-in role permission have the permission to perform GRANT and REVOKE operations on built-in roles. Example:

```
GRANT gs_role_signal_backend TO user1;  
REVOKE gs_role_signal_backend FROM user1;
```

2.1.6 Schemas

Schemas allow multiple users to use the same database without interference. In this way, database objects can be organized into logical groups that are easy to manage, and third-party applications can be added to corresponding schemas without causing conflicts.

Each database has one or more schemas. Each schema contains tables and other types of objects. When a database is created, a public schema named **public** is created by default, and all users have the USAGE permission on this schema. In addition, each database has a pg_catalog schema, which contains system catalogs and all built-in data types, functions, and operators. Only a system administrator and the initial user can create functions, stored procedures, and synonyms under the public and pg_catalog schemas. Other users cannot create these objects even if they are granted with the CREATE permission on the public and pg_catalog schemas. You can group database objects by schema. A schema is similar to an OS directory but cannot be nested. By default, only the initial user can create objects in pg_catalog.

The same database object name can be used in different schemas of the same database without causing conflicts. For example, both a_schema and b_schema can contain a table named **mytable**. Users with required permissions can access objects across multiple schemas of the same database.

When you run the **CREATE USER** command to create a user, the system creates a schema with the same name as the user in the database where the command is executed.

Database objects are generally created in the first schema in a database search path. For details about the first schema and how to change the schema order, see [Search Path](#).

Creating, Modifying, and Deleting Schemas

- To create a schema, use **CREATE SCHEMA**. By default, the initial user and system administrators can create schemas. Other users can create schemas in the database only when they have the CREATE permission on the database. For details about how to grant the permission, see the syntax in [GRANT](#).
- To change the name or owner of a schema, use **ALTER SCHEMA**. The schema owner can change a schema.
- To delete a schema and its objects, use **DROP SCHEMA**. The schema owner can delete a schema.
- To create a table in a schema, use the *schema_name.table_name* format to specify the table. If *schema_name* is not specified, the table will be created in the first schema in [search path](#).

- To view the owner of a schema, perform the join query on the system catalogs PG_NAMESPACE and PG_USER. Replace *schema_name* in the statement with the name of the schema to be queried.

```
gaussdb=# SELECT s.nspname,u.username AS nspowner FROM pg_namespace s, pg_user u WHERE nspname='schema_name' AND s.nspowner = u.usesysid;
```
- To view a list of all schemas, query the system catalog PG_NAMESPACE.

```
gaussdb=# SELECT * FROM pg_namespace;
```
- To view a list of tables in a schema, query the system catalog PG_TABLES. For example, the following query will return a table list from PG_CATALOG in the schema:

```
gaussdb=# SELECT distinct(tablename),schemaname from pg_tables where schemaname = 'pg_catalog';
```

Search Path

A search path is defined in the GUC parameter **search_path**. The parameter value is a list of schema names separated by commas (,). If no target schema is specified during object creation, the object will be added to the first schema listed in the search path. If there are objects with the same name across different schemas and no schema is specified for an object query, the object will be returned from the first schema containing the object in the search path.

- To view the current search path, use **SHOW**.

```
gaussdb=# SHOW SEARCH_PATH;
search_path
-----
"$user",public
(1 row)
```

The default value of **search_path** is "*\$user*",**public**. *\$user* indicates the name of the schema with the same name as the current session user. If the schema does not exist, *\$user* will be ignored. By default, after a user connects to a database that has schemas with the same name, objects will be added to all the schemas. If there are no such schemas, objects will be added only to the public schema.

- To change the default schema of the current session, run the **SET** command.

Set the search path to **myschema, public** (**myschema** will be searched first).

```
gaussdb=# SET SEARCH_PATH TO myschema, public;
SET
```

2.1.7 User Permissions

- To grant permissions for an object to a user, use **GRANT**.

When permissions for a table or view in a schema are granted to a user or role, the USAGE permission of the schema must be granted together. Otherwise, the user or role can only see these objects but cannot access them.

In the following example, permissions for the schema **tpcds** are first granted to user **joe**, and then the SELECT permission for the **tpcds.web_returns** table is also granted:

```
gaussdb=# GRANT USAGE ON SCHEMA tpcds TO joe;
gaussdb=# GRANT SELECT ON TABLE tpcds.web_returns to joe;
```

- Grant a role to a user to allow the user to inherit the object permissions of the role.
 - a. Create a role.
Create a role **lily** and grant the system permission **CREATEDB** to the role.

```
gaussdb=# CREATE ROLE lily WITH CREATEDB PASSWORD '*****';
```

- b. To grant object permissions to a role, use **GRANT**.

For example, first grant permissions for the schema `tpcds` to the role `lily`, and then grant the `SELECT` permission of the `tpcds.web_returns` table to `lily`.

```
gaussdb=# GRANT USAGE ON SCHEMA tpcds TO lily;
gaussdb=# GRANT SELECT ON TABLE tpcds.web_returns TO lily;
```

- c. Grant the role permissions to a user.

```
gaussdb=# GRANT lily TO joe;
```

NOTE

When the permissions of a role are granted to a user, the attributes of the role are not transferred together.

- To revoke user permissions, use **REVOKE**.

2.1.8 Row-Level Security

The row-level security feature enables database access control to be accurate to each row of data tables. In this way, the same SQL query may return different results for different users.

You can create a row-level security policy for a data table. The policy defines an expression that takes effect only for specific database users and SQL operations. When a database user accesses the data table, if an SQL statement meets the specified row-level security policies of the data table, the expressions that meet the specified condition will be combined by using `AND` or `OR` based on the attribute type (`PERMISSIVE` | `RESTRICTIVE`) and applied to the execution plan in the query optimization phase.

Row-level security policy is used to control the visibility of row-level data in tables. By predefining filters for data tables, the expressions that meet the specified condition can be applied to execution plans in the query optimization phase, which will affect the final execution result. Currently, the SQL statements that can be affected include `SELECT`, `UPDATE`, and `DELETE`.

Scenario 1: A table summarizes the data of different users. Users can view only their own data.

```
-- Create users alice, bob, and peter.
gaussdb=# CREATE USER alice PASSWORD '*****';
gaussdb=# CREATE USER bob PASSWORD '*****';
gaussdb=# CREATE USER peter PASSWORD '*****';

-- Create the all_data table that contains user information.
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

-- Insert data into the data table.
gaussdb=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
gaussdb=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
gaussdb=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

-- Grant the read permission for the all_data table to users alice, bob, and peter.
gaussdb=# GRANT SELECT ON all_data TO alice, bob, peter;

-- Enable row-level security policy.
gaussdb=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

-- Create a row-level security policy to specify that the current user can view only their own data.
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);
```

```
-- View table details.
gaussdb=# \d+ all_data
          Table "public.all_data"
Column |          Type          | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
id     | integer                |           |         |              |
role   | character varying(100) |           | extended |              |
data   | character varying(100) |           | extended |              |
Row Level Security Policies:
  POLICY "all_data_rls" FOR ALL
  TO public
  USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Switch to user alice and run SELECT * FROM public.all_data.
gaussdb=# SELECT * FROM public.all_data;
id | role | data
---+---+---
 1 | alice | alice data
(1 row)

gaussdb=# EXPLAIN(COSTS OFF) SELECT * FROM public.all_data;
          QUERY PLAN
-----
Streaming (type: GATHER)
  Node/s: All datanodes
  -> Seq Scan on all_data
      Filter: ((role)::name = 'alice'::name)
Notice: This query is influenced by row level security feature
(5 rows)

-- Switch to user peter and run SELECT * FROM public.all_data.
gaussdb=# SELECT * FROM public.all_data;
id | role | data
---+---+---
 3 | peter | peter data
(1 row)

gaussdb=# EXPLAIN(COSTS OFF) SELECT * FROM public.all_data;
          QUERY PLAN
-----
Streaming (type: GATHER)
  Node/s: All datanodes
  -> Seq Scan on all_data
      Filter: ((role)::name = 'peter'::name)
Notice: This query is influenced by row level security feature
(5 rows)
```

NOTICE

PG_STATISTIC and PG_STATISTIC_EXT store sensitive information about statistical objects, such as high-frequency MCVs. If the permission to query the two system catalogs is granted to a common user after the row-level security policy is created, the common user can still access the two system catalogs to obtain sensitive information in the statistical objects.

PG_STATS, PG_EXT_STATS, and PG_GTT_STATS are statistics views and require that users have the SELECT permission on columns. If a common user has the SELECT permission and row-level security is configured on the table to query, the common user may retrieve some data that should not be retrieved but happens to be MCVs in the statistics views (not a whole row but a certain column value because the statistics are collected and calculated by column.)

3 Database Quick Start

3.1 Connecting to a Database

Client tools used for connecting to a database include **gsql** and APIs (such as **ODBC** and **JDBC**).

- **gsql** is a client tool provided by GaussDB. As described in [Using gsql to Connect to a Database](#), **gsql** is used to enter, edit, and run SQL statements in an interactive manner.
- As described in [APIs](#), you can use standard database APIs, such as **ODBC** and **JDBC**, to develop GaussDB-based applications.

3.1.1 Using gsql to Connect to a Database

gsql is a client tool provided by GaussDB. You can use **gsql** to connect to the database and then enter, edit, and execute SQL statements in an interactive manner. For details about the connection mode, see [Using gsql to Connect to an Instance](#).

NOTICE

For distributed instances, you can use a client tool to connect to a database through any CN. Before connection, you must obtain the IP address and port number of the server where the CN is deployed. The client tool can access the database by connecting to any CN. Do not connect to a database through a DN when services are running properly.

3.1.2 APIs

You can use standard database APIs, such as **ODBC** and **JDBC**, to develop GaussDB-based applications.

Supported APIs

Each application is an independent GaussDB development project. APIs alleviate applications from directly operating in databases, and enhance the database

portability, extensibility, and maintainability. [Table 3-1](#) lists the APIs supported by GaussDB and the download addresses.

Table 3-1 Database APIs

API	How to Obtain
ODBC	<ul style="list-style-type: none">Linux: Driver: GaussDB-Kernel_Database version number_OS information_64bit_Odbc.tar.gz unixODBC source code package: https://gitee.com/src-openeuler/unixODBC/blob/openEuler-22.03-LTS-SP1/unixODBC-2.3.7.tar.gzWindows: Driver: GaussDB-Kernel_Database version number_Windows_Odbc.tar.gz
JDBC	<ul style="list-style-type: none">Driver: GaussDB-Kernel_Database version number_OS information_64bit-Jdbc.tar.gzDriver: org.postgresql.Driver

The JDBC and ODBC APIs are used to connect to the database remotely. Therefore, you need to configure a remote connection in GaussDB. Contact an administrator for related operations.

For details about more APIs, see [Application Development Guide](#).

3.2 Operating a Database

This section explains how to use databases, including creating database accounts, databases, and tables, inserting data to tables, and querying data in tables.

3.2.1 Creating a Database Account

Only administrators that are created during the cluster installation can access the initial database by default. You can also create other database users.

```
gaussdb=# CREATE USER joe WITH PASSWORD '*****';
```

If the following information is displayed, the creation is successful:

```
CREATE ROLE
```

In this case, you have created a user account named **joe**, and the user password is *********. Set user **joe** as a system administrator.

```
gaussdb=# GRANT ALL PRIVILEGES TO joe;
```

Run the **GRANT** command to set related permissions. For details, see [GRANT](#).

NOTE

For more information about database users, see [Users and Permissions](#).

3.2.2 Creating and Managing Databases

Prerequisites

Only database system administrators or users granted with database creation permissions can create a database. For details about how to grant database creation permissions to a user, see [Users and Permissions](#).

Context

- GaussDB has two default template databases **template0** and **template1** and a default user database **postgres**.
- CREATE DATABASE creates a database by copying a template database. Only **template0** can be copied. Do not use a client or other methods to connect to or operate the two template databases.

NOTE

- The template database does not contain any user table. You can view the attributes of the template database in the PG_DATABASE system catalog.
- The **template0** template does not allow user connections. Only the initial user of the database and the system administrator can connect to **template1**.
- A database system consists of multiple databases. A client can connect to only one database at a time. Currently, cross-database query or cross-database transaction is not supported.
- If multiple databases exist in the database cluster, you can use the **-d** parameter of the client tool to specify the target database for login. Alternatively, you can run the **\c** command to switch the database after the client program logs in to the database.

Precautions

Assume that the database encoding is SQL_ASCII. (You can run the **show server_encoding** command to query the encoding used for storing data in the current database.) If the database object name contains multi-byte characters (such as Chinese) or if the object name length exceeds the allowed maximum (63 bytes), the database truncates the last byte (not the last character) of the object name. In this case, half characters may appear.

To resolve this problem, you need to:

- Ensure that the name of the data object does not exceed the maximum length.
- Use a proper coded character set, such as UTF-8, as the default database storage code set (server_encoding).
- Exclude multi-byte characters from object names.
- If you fail to delete an object whose name is truncated mistakenly, specify its original name to delete it, or manually delete it from the corresponding system catalog on each node.

Procedure

Step 1 Create a database named **db_tpcds**.

```
gaussdb=# CREATE DATABASE db_tpcds;  
CREATE DATABASE
```

NOTE

- Database names must comply with the general naming convention rules of SQL identifiers. The current role automatically becomes the owner of this new database.
- If a database system is used to support independent users and projects, store them in different databases.
- If the projects or users are associated with each other and share resources, store them in one database. However, you can divide them into different schemas. A schema is a logical structure, and the access permission for a schema is controlled by the permission system module.
- A database name contains a maximum of 63 bytes and the excessive bytes at the end of the name will be truncated by the server. You are advised to specify a database name no longer than 63 bytes when you create a database.

- New databases are created in the `pg_default` tablespace by default. Specify another tablespace.

```
gaussdb=# CREATE DATABASE db_tpcds WITH TABLESPACE = hr_local;  
CREATE DATABASE
```

hr_local indicates the tablespace name. For details about how to create a tablespace, see [Creating and Managing Tablespaces](#).

- After creating the **db_tpcds** database, you can perform other operations in the default **postgres** database. Alternatively, you can perform the following operations to exit the **postgres** database, connect to the **db_tpcds** database as a new user, and perform operations such as creating tables.

```
gaussdb=# \q  
gsq! -d db_tpcds -p 8000 -U joe  
Password for user joe:  
gsq!((GaussDB Kernel 503.0.XXX build f521c606) compiled at 2021-09-16 14:55:22 commit 2935  
last mr 6385 release)  
Non-SSL connection (SSL connection is recommended when requiring high-security)  
Type "help" for help.  
  
db_tpcds=>
```

Step 2 View databases.

- Run the `\l` meta-command to view the database list of the database system.

```
gaussdb=# \l
```

- Query the database list in the `pg_database` system catalog.

```
gaussdb=# SELECT datname FROM pg_database;
```

Step 3 Modify the database.

You can modify database configuration such as the database owner, name, and default settings.

- Rename the database.

```
gaussdb=# ALTER DATABASE db_tpcds RENAME TO human_tpcds;  
ALTER DATABASE
```

NOTE

After setting the parameters, you need to manually run the **CLEAN CONNECTION** command to clear the old connections. Otherwise, the parameter values between nodes may be inconsistent.

Step 4 Drop the database.

You can run the **DROP DATABASE** command to delete a database. This command deletes the system directory in the database, as well as the database directory on the disk that stores data. Only the database owner or system administrator can delete a database. A database accessed by users cannot be deleted. You need to connect to another database before deleting this database.

Drop the database.

```
gaussdb=# DROP DATABASE human_tpcds;  
DROP DATABASE
```

----End

3.2.3 Creating and Managing Tablespaces

Context

The administrator can use tablespaces to control the layout of disks where a database is installed. The advantages are as follows:

- If the disk partition or tablespace initially allocated to the database is full and the space cannot be logically extended, you can create and use tablespaces in other partitions until the disk space is reconfigured.
- Tablespaces allow the administrator to distribute data based on a schema of database objects, improving system performance.
 - A frequently used index can be stored in a disk having stable performance and high computing speed, such as a solid-state device.
 - A table that stores archived data and is rarely used or has low performance requirements can be placed in a disk with a slow computing speed.
- You can use tablespaces to limit the disk space of databases. If a tablespace shares a partition with other tablespaces, the tablespace will never occupy the space allocated to other tablespaces.
- You can use tablespaces to manage the disk space occupied by a database. The database will switch to the read-only mode when the usage of the disk occupied by the database reaches 90%. Once the disk usage drops below 90%, the database will be switched to the read-write mode. The automatic disk check function of the CM component is enabled by default. If the function is disabled, contact the administrator to enable it.
- Each tablespace corresponds to a file system directory. You can run the following command to create a tablespace corresponding to **/pg_location/mount1/path1** and specify the maximum available space to 500 GB:

```
-- Create a tablespace.
```

```
gaussdb=# CREATE TABLESPACE ds_location1 LOCATION '/pg_location/mount1/path1' MAXSIZE  
'500G';
```

If **MAXSIZE** is used to manage tablespace quotas, the concurrent insertion performance may deteriorate by about 30%. **MAXSIZE** specifies the maximum quota for each DN. The difference between the actual tablespace capacity of each DN and the specified quota should be within 500 MB. Determine whether to set a tablespace to its maximum size as required.

GaussDB provides two tablespaces: pg_default and pg_global.

- Default tablespace `pg_default`: stores non-shared system catalogs, user tables, user table indexes, temporary tables, temporary table indexes, and internal temporary tables. The corresponding storage directory is the base directory in the instance data directory.
- Shared tablespace `pg_global`: stores shared system catalogs. The corresponding storage directory is the base directory in the global data directory.

Precautions:

- You are advised not to use user-defined tablespaces in the Huawei Cloud scenario.

This is because user-defined tablespaces are usually used with storage media other than the main storage (storage device where the default tablespace is located, such as a disk) to isolate I/O resources that can be used by different services. However, in the Huawei Cloud scenario, storage devices use standard configurations and do not have other available storage media. If the user-defined tablespaces are not properly used, the system cannot run stably for a long time and the overall performance is affected. Therefore, you are advised to use the default tablespace.

Procedure

- Create a tablespace.

- a. Create user **jack**.

```
gaussdb=# CREATE USER jack IDENTIFIED BY '*****';
```

If the following information is displayed, the creation is successful:

```
CREATE ROLE
```

- b. Create a tablespace.

```
gaussdb=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'my_tablespace/tablesace1';
```

If the following information is displayed, the creation is successful:

```
CREATE TABLESPACE
```

fastspace is the new tablespace, and *CN or DN data directory* **pg_location/my_tablespace/tablesace1** is an empty directory on which users have read and write permissions.

- c. Grant the permission of accessing the **fastspace** tablespace to user **jack** as a database system administrator.

```
gaussdb=# GRANT CREATE ON TABLESPACE fastspace TO jack;
```

If the following information is displayed, the grant operation is successful:

```
GRANT
```

- Create an object in a tablespace.

If you have the CREATE permission on the tablespace, you can create database objects in the tablespace, such as tables and indexes.

Take creating a table as an example:

- Method 1: Create a table in a specified tablespace.

```
gaussdb=# CREATE TABLE foo(i int) TABLESPACE fastspace;
```

If the following information is displayed, the creation is successful:

```
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'i' as the distribution column by default.
```

```
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
```

```
CREATE TABLE
```

- Method 2: Run **SET default_tablespace** to set the default tablespace and then create a table.

```
gaussdb=# SET default_tablespace = 'fastspace';
SET
gaussdb=# CREATE TABLE foo2(i int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'i' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
```

In this example, **fastspace** is the default tablespace, and **foo2** is the created table.

- Query tablespaces.
 - Method 1: Check the pg_tablespace system catalog. View all tablespaces defined by the system and users.

```
gaussdb=# SELECT spcname FROM pg_tablespace;
```
 - Method 2: Run a gsql meta-command to query tablespaces.

```
gaussdb=# \db
```
- Query the tablespace usage.

- a. Query the current usage of a tablespace.

```
gaussdb=# SELECT PG_TABLESPACE_SIZE('fastspace');
```

Information similar to the following is displayed:

```
pg_tablespace_size
-----
                2146304
(1 row)
```

2146304 is the size of the tablespace, and its unit is byte.

- b. Calculate the tablespace usage.

Tablespace usage = Value of **PG_TABLESPACE_SIZE**/Size of the disk where the tablespace resides

- Modify a tablespace.

Rename tablespace **fastspace** to **fspace**.

```
gaussdb=# ALTER TABLESPACE fastspace RENAME TO fspace;
ALTER TABLESPACE
```

- Delete a tablespace and related data.

- Delete user **jack**.

```
gaussdb=# DROP USER jack CASCADE;
DROP ROLE
```

- Delete tables **foo** and **foo2**.

```
gaussdb=# DROP TABLE foo;
gaussdb=# DROP TABLE foo2;
```

If the following information is displayed, the operation is successful:

```
DROP TABLE
```

- Delete tablespace **fspace**.

```
gaussdb=# DROP TABLESPACE fspace;
DROP TABLESPACE
```

NOTE

Only the tablespace owner or system administrators can delete a tablespace.

3.2.4 Creating and Managing Tables

3.2.4.1 Creating a Table

Context

A table is created in a database and can be stored in different databases. Tables under different schemas in a database can have the same name.

For details about how to design a table suitable for services, see [Best Practices of Table Design](#).

Creating a Table

Create a table.

```
gaussdb=# CREATE TABLE customer_t1
(
  c_customer_sk      integer,
  c_customer_id     char(5),
  c_first_name      char(6),
  c_last_name       char(8)
)
distribute by hash (c_last_name);
```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

c_customer_sk, **c_customer_id**, **c_first_name**, and **c_last_name** are the column names of the table. **integer**, **char(5)**, **char(6)**, and **char(8)** are column name types.

NOTE

- By default, new database objects are created in the \$user schema. For more details about schemas, see [Creating and Managing Schemas](#).
- In addition to the created tables, a database contains many system catalogs. These system catalogs contain cluster installation information and information about various queries and processes in GaussDB. You can collect information about the database by querying system catalogs. For details, see [Querying System Catalogs](#).
- For more details about how to create a table, see [CREATE TABLE](#).

3.2.4.2 Inserting Data to a Table

A new table contains no data. You need to insert data to the table before using it. This section describes how to insert a row or multiple rows of data by running the **INSERT** command and to insert data from a specified table. Contact an administrator if a large amount of data needs to be imported.

Context

In case of different character sets used on the client and server, the length of a character on both ends may differ. A string entered on the client will be processed on the server based on the server's character set, and therefore the result may differ from the expected result.

Table 3-2 Comparison of outputs varying depending on character sets on the client and server

Procedure	Same Encoding on the Server and Client	Different Encoding on the Server and Client
No operations are performed on the string while it is saved and read.	Your expected result is returned.	Your expected result is returned (only if the encoding on the client remains unchanged).
Operations (such as executing string functions) are performed on the string while it is saved and read.	Your expected result is returned.	The result may differ from the expected result, depending on the operations performed on the string.
An ultra long string is truncated while it is saved.	Your expected result is returned.	If the character sets used on the client and server have different character length, the result may differ from the expected result.

More than one of the preceding operations may be performed on a string. For example, if the character sets of the client and server are different and the string is truncated after being processed, the result will also be unexpected. For details, see [Table 3-3](#).

 **NOTE**

Ultra long strings are truncated only if **DBCMPATIBILITY** is set to **TD** (compatible with Teradata) and the GUC parameter **td_compatible_truncation** is set to **on**.

Create **table1** and **table2** to be used in the examples.

```
gaussdb=# CREATE TABLE table1(id int, a char(6), b varchar(6),c varchar(6));
gaussdb=# CREATE TABLE table2(id int, a char(20), b varchar(20),c varchar(20));
```

Table 3-3 Example

No.	Character Set on the Server	Character Set on the Client	Automatic Truncation Enabled	Example	Result	Description
1	SQL_ASCII	UTF8	Yes	gaussdb=# INSERT INTO table1 VALUES(1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78'));	id a b c ----+----- +-----+----- 1 87 87 87	A string is reversed on the server and then truncated. Because character sets used by the server and client are different, character A is displayed in multiple bytes on the server and the result is unexpected.
2	SQL_ASCII	UTF8	Yes	gaussdb=# INSERT INTO table1 VALUES(2,reverse('123A78'),reverse('123A78'),reverse('123A78'));	id a b c ----+----- +-----+----- 2 873 873 873	A string is reversed and then automatically truncated. Therefore, the result is unexpected.
3	SQL_ASCII	UTF8	Yes	gaussdb=# INSERT INTO table1 VALUES(3,'87A123','87A123','87A123');	id a b c ----+----- +-----+----- 3 87A1 87A1 87A1	The column length in the string type is an integer multiple of the length in client character encoding. Therefore, the result is as expected after truncation.

No.	Character Set on the Server	Character Set on the Client	Automatic Truncation Enabled	Example	Result	Description
4	SQL_ASCII	UTF8	No	<pre>gaussdb=# INSERT INTO table2 VALUES(1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')); gaussdb=# INSERT INTO table2 VALUES(2,reverse('123A78'),reverse('123A78'),reverse('123A78')) ;</pre>	<pre>id a b c ---- +-----+ --+----- +----- 1 87 321 87 321 87 321 2 87321 87321 87321</pre>	Similar to the first example, multi-byte characters are different from the original characters after being reversed.

Procedure

You need to create a table before inserting data to it. For details about how to create a table, see [Creating and Managing Tables](#).

- Insert a row to table **customer_t1**.

Data values are arranged in the same order as the columns in the table and are separated by commas (.). Generally, column values are text values (constants), but scalar expressions are also allowed.

```
gaussdb=# INSERT INTO customer_t1(c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', 'Grace');
```

If you know the sequence of the columns in the table, you can obtain the same result without listing these columns. For example, the following command generates the same result as the preceding command:

```
gaussdb=# INSERT INTO customer_t1 VALUES (3769, 'hello', 'Grace');
```

If you do not know some of the column values, you can omit them. If no value is specified for a column, the column is set to the default value.

Example:

```
gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_first_name) VALUES (3769, 'Grace');
```

```
gaussdb=# INSERT INTO customer_t1 VALUES (3769, 'hello');
```

You can also specify the default value of a column or row.

```
gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', DEFAULT);
```

```
gaussdb=# INSERT INTO customer_t1 DEFAULT VALUES;
```

- Insert multiple rows to a table.

```
gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
(6885, 'maps', 'Joes'),
(4321, 'tpcds', 'Lily'),
(9527, 'world', 'James');
```

You can also insert multiple rows by running the command for inserting one row for multiple times. However, you are advised to run this command to improve efficiency.

- Assume that you have created a backup table **customer_t2** for table **customer_t1**. To insert data from **customer_t1** to **customer_t2**, run the following commands:

```
gaussdb=# CREATE TABLE customer_t2
(
  c_customer_sk      integer,
  c_customer_id      char(5),
  c_first_name       char(6),
  c_last_name        char(8)
);

gaussdb=# INSERT INTO customer_t2 SELECT * FROM customer_t1;
```

NOTE

If implicit conversion is not implemented between the column data types of the specified table and those of the current table, the two tables must have the same column data types when data is inserted from the specified table to the current table.

- Delete a backup table.
gaussdb=# DROP TABLE customer_t2 CASCADE;

NOTE

If the table to be deleted is dependent on other tables, you need to delete the corresponding tables first.

3.2.4.3 Updating Data in a Table

Existing data in a database can be updated. You can update one row, all rows, or specified rows of data, or update data in a single column without affecting the data in the other columns.

The following information is required when the UPDATE statement is used to update rows:

- Table name and column name of the data to be updated
- New column value
- Rows to be updated

Generally, in SQL, no unique ID is specified for a row of data. Therefore, it is impossible to directly specify the rows of the data to be updated. However, you can specify the conditions that must be met by the rows to be updated. If a table contains primary keys, you can specify a row using the primary keys.

For details about how to create a table and insert data to it, see [Creating a Table](#) and [Inserting Data to a Table](#).

c_customer_sk in the table **customer_t1** must be changed from **9527** to **9876**.

```
gaussdb=# UPDATE customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;
```

You can use a schema to modify the table name. If no such modifier is specified, the table is located based on the default schema path. SET is followed by the column and the new column value. The new value can be a constant or an expression.

For example, increase all the values in the **c_customer_sk** column by 100.

```
gaussdb=# UPDATE customer_t1 SET c_customer_sk = c_customer_sk + 100;
```

This statement does not contain the WHERE clause, and therefore all rows are updated. If the statement contains the WHERE clause, only the rows matching the clause are updated.

In the SET clause, the equal sign (=) indicates value setting. In the WHERE clause, the equal sign indicates comparison. A WHERE condition does not have to be an equality comparison, and can be another operator.

You can use an UPDATE statement to update multiple columns by specifying multiple values in the SET clause, for example:

```
gaussdb=# UPDATE customer_t1 SET c_customer_id = 'Admin', c_first_name = 'Local' WHERE  
c_customer_sk = 4421;
```

After data has been updated or deleted in batches, a large number of deletion markers are generated in the data file. During query, data with these deletion markers needs to be scanned as well. In this case, a large amount of data with deletion marks can greatly affect the query performance after batch updates or deletions. If data needs to be updated or deleted in batches frequently, you are advised to periodically run **VACUUM FULL** so as to ensure the query performance.

3.2.4.4 Viewing Data

- Run the following command to query information about all tables in a database in the system catalog **pg_tables**:
gaussdb=# SELECT * FROM pg_tables;
- Run the **\d+** command of the **gsql** tool to query table attributes:
gaussdb=# \d+ customer_t1;
- Run the following command to query the data volume of table **customer_t1**:
gaussdb=# SELECT count(*) FROM customer_t1;
- Run the following command to query all data in the table **customer_t1**:
gaussdb=# SELECT * FROM customer_t1;
- Run the following command to query only the data in the column **c_customer_sk**:
gaussdb=# SELECT c_customer_sk FROM customer_t1;
- Run the following command to filter repeated data in the column **c_customer_sk**:
gaussdb=# SELECT DISTINCT(c_customer_sk) FROM customer_t1;
- Run the following command to query all data whose column **c_customer_sk** is **3869**:
gaussdb=# SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;
- Run the following command to collate data based on the column **c_customer_sk**:
gaussdb=# SELECT * FROM customer_t1 ORDER BY c_customer_sk;

3.2.4.5 Deleting Data from a Table

Outdated data may need to be deleted when tables are used. Data can be deleted from tables only by row.

In SQL, you can only access and delete an independent row by declaring conditions. If a table has a primary key, you can use it to specify a row. You can delete several rows that match the specified condition or delete all the rows from a table.

For example, delete all the rows whose `c_customer_sk` column is **3869** from the table `customer_t1`.

```
gaussdb=# DELETE FROM customer_t1 WHERE c_customer_sk = 3869;
```

To delete all rows from the table, run either of the following commands:

```
gaussdb=# DELETE FROM customer_t1;
```

Or

```
gaussdb=# TRUNCATE TABLE customer_t1;
```

NOTE

If you need to delete an entire table, you are advised to use the TRUNCATE statement rather than DELETE.

Delete the created table.

```
gaussdb=# DROP TABLE customer_t1;
```

3.2.5 Querying System Catalogs

In addition to the created tables, a database contains many system catalogs. These system catalogs contain cluster installation information and information about various queries and processes in GaussDB. You can obtain information about the database by querying system catalogs.

In [System Catalogs and System Views](#), the description about each table specifies whether the table is visible to all users or only the initial user. To query tables that are visible only to the initial user, log in as the initial user.

GaussDB provides the following types of system catalogs and views:

- PostgreSQL-compatible system catalogs and views
These system catalogs and views have the prefix **PG**.
- New system catalogs and views of GaussDB
These system catalogs and views have the prefix **GS**.
- Oracle-compatible system catalogs and views
These system catalogs and views have the prefix **ALL**, **DBA**, **USER**, or **PV**.

Querying Database Tables

Create the following tables in the public schema:

```
gaussdb=# CREATE TABLE public.search_table_t1(a int) distribute by hash(a);  
CREATE TABLE  
gaussdb=# CREATE TABLE public.search_table_t2(b int) distribute by hash(b);  
CREATE TABLE  
gaussdb=# CREATE TABLE public.search_table_t3(c int) distribute by hash(c);  
CREATE TABLE  
gaussdb=# CREATE TABLE public.search_table_t4(d int) distribute by hash(d);  
CREATE TABLE  
gaussdb=# CREATE TABLE public.search_table_t5(e int) distribute by hash(e);  
CREATE TABLE
```

In the PG_TABLES system catalog, view the tables prefixed with search_table in the public schema.

```
gaussdb=# SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public' AND  
TABLENAME LIKE 'search_table%';
```

The result is as follows:

```

tablename
-----
search_table_t1
search_table_t2
search_table_t3
search_table_t4
search_table_t5
(5 rows)

```

Viewing Database Users

You can use PG_USER to view the list of all users in the database, and view the user ID (**USESYSID**) and permissions.

```

SELECT * FROM pg_user;
username | usesysid | usecreatedb | usesuper | usecatupd | userepl | passwd | valbegin | valuntil |
respool  | parent  | spacelimit | useconfig | nodegroup | tempspacelimit | spillspacelimit
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----
roach | 10 | t | t | t | t | ***** | | | default_pool |
0 | | | | | | | | |
(1 row)

```

Viewing and Stopping the Running Query Statements

You can view the running query statements in the **PG_STAT_ACTIVITY** view. You can use the following methods:

Step 1 Set the parameter **track_activities** to **on**.

```
SET track_activities = on;
```

The database obtains the running information about active queries only if the parameter is set to **on**.

Step 2 View the running query statements. View the database names, users performing queries, query status, and the corresponding PID which are connected to the running query statements.

```

SELECT datname, username, state,pid FROM pg_stat_activity;
datname | username | state | pid
-----+-----+-----+-----
testdb | Ruby | active | 140298793514752
testdb | Ruby | active | 140298718004992
testdb | Ruby | idle | 140298650908416
testdb | Ruby | idle | 140298625742592
testdb | omm | active | 140298575406848
(5 rows)

```

If the **state** column is **idle**, the connection is idle and requires a user to enter a command.

View the query statements that are not in the idle state.

```
SELECT datname, username, state, pid FROM pg_stat_activity WHERE state != 'idle';
```

Step 3 To cancel queries that have been running for a long time, use the **PG_TERMINATE_BACKEND** function to end sessions based on the thread ID (corresponding to the PID in **Step 2**).

```
SELECT PG_TERMINATE_BACKEND(140298793514752);
```

If the following information is displayed, the session is successfully terminated:

```
PG_TERMINATE_BACKEND
-----
t
(1 row)
```

If the following information is displayed, a user has terminated the current session:

```
FATAL: terminating connection due to administrator command
FATAL: terminating connection due to administrator command
```

NOTE

1. When an initial user uses the PG_TERMINATE_BACKEND function to terminate the backend threads of active sessions, the gsql client is reconnected automatically rather than be logged out. The message "The connection to the server was lost. Attempting reset: Succeeded." is returned. If non-initial users do this operation, the message "The connection to the server was lost. Attempting reset: Failed." is returned. This is because only initial users can log in to the system in password-free mode.
2. If the PG_TERMINATE_BACKEND function is used to end inactive backend threads, and the thread pool is enabled, idle sessions do not have thread IDs and cannot be terminated. In non-thread pool mode, terminated sessions are not automatically reconnected.

----End

3.2.6 Other Operations

3.2.6.1 Creating and Managing Schemas

Context

Schemas allow multiple users to use the same database without interference. In this way, database objects can be organized into logical groups that are easy to manage, and third-party applications can be added to corresponding schemas without causing conflicts. Schema management involves creating a schema, using a schema, deleting a schema, setting a search path for a schema, and setting schema permissions.

Precautions

- A database cluster can have one or more databases. Users and user groups are shared within entire cluster, but their data is exclusive. Any user who has connected to a server can access only the database specified in the connection request.
- A database can have one or more schemas, and a schema can contain tables and other data objects, such as data types, functions, and operators. One object name can be used in different schemas. For example, both **schema1** and **schema2** can have a table named **mytable**.
- Different from databases, schemas are not isolated. You can access the objects in a schema of the connected database if you have schema permissions. To manage schema permissions, you need to have knowledge about database permissions.
- A schema named with the **PG_** prefix cannot be created because this type of schema is reserved for the database system.

- Each time a new user is created, the system creates a schema with the same name for the new user in the current database. In other databases, such a schema needs to be manually created.
- To reference a table that is not modified with a schema name, the system uses **search_path** to find the schema that the table belongs to. `pg_temp` and `pg_catalog` are always the first two schemas to be searched no matter whether or how they are specified in **search_path**. **search_path** is a schema name list, and the first table detected in it is the target table. If no target table is found, an error will be reported. (If a table exists but the schema it belongs to is not listed in **search_path**, the search fails as well.) The first schema in **search_path** is called "current schema". This schema is the first one to be searched. If no schema name is declared, newly created database objects are saved in this schema by default.
- Each database has a `pg_catalog` schema, which contains system catalogs and all built-in data types, functions, and operators. `pg_catalog` is a part of the search path and has the second highest search priority. It is searched after the schema of temporary tables and before other schemas specified in **search_path**. This search order ensures that database built-in objects can be found. To use a custom object that has the same name as a built-in object, you can specify the schema of the custom object.

Procedure

- Create a schema.
 - Create a schema.

```
gaussdb=# CREATE SCHEMA myschema;
```

If the following information is displayed, the schema named **myschema** is successfully created:

```
CREATE SCHEMA
```

To create or access an object in the schema, specify the complete object name, which consists of the schema name and the object names separated by periods (.). Example: **myschema.table**.
 - Create a schema and specify the owner.

```
gaussdb=# CREATE SCHEMA myschema AUTHORIZATION omm;
```

If the following information is displayed, the **myschema** schema that belongs to the **omm** user is created successfully:

```
CREATE SCHEMA
```
- Use a schema.

If you want to create or access an object in a specified schema, the object name must contain the schema name. To be specific, the name consists of a schema name and an object name, which are separated by a dot (.).

 - Create the **mytable** table in **myschema**.

```
gaussdb=# CREATE TABLE myschema.mytable(id int, name varchar(20));
```

```
CREATE TABLE
```

To specify the location of an object, the object name must contain the schema name.
 - Query all data of the **mytable** table in **myschema**.

```
gaussdb=# SELECT * FROM myschema.mytable;
```

```
id | name
----+-----
(0 rows)
```

- View the search path of a schema.

You can set **search_path** to specify the sequence of schemas in which objects are searched. The first schema listed in the search path will become the default schema. If no schema is specified during object creation, the object will be created in the default schema.

- View the search path.

```
gaussdb=# SHOW SEARCH_PATH;
search_path
-----
"$user",public
(1 row)
```

- Set the search path to **myschema, public** (**myschema** will be searched first).

```
gaussdb=# SET SEARCH_PATH TO myschema, public;
SET
```

- Set permissions for a schema.

By default, a user can only access database objects in their own schema. Only after a user is granted with the usage permission for a schema by the schema owner, the user can access the objects in the schema.

By granting the CREATE permission for a schema to a user, the user can create objects in this schema. By default, all roles have the USAGE permission in the public schema, but common users do not have the CREATE permission in the public schema. It is insecure for a common user to connect to a specified database and create objects in its public schema. If the common user has the CREATE permission on the public schema, it is advised to:

- Revoke **PUBLIC**'s permission to create objects in the public schema.

```
gaussdb=# REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE
```

- View the current schema.

```
gaussdb=# SELECT current_schema();
current_schema
-----
myschema
(1 row)
```

- Create user **jack** and grant the usage permission for **myschema** to the user.

```
gaussdb=# CREATE USER jack IDENTIFIED BY '*****';
CREATE ROLE
gaussdb=# GRANT USAGE ON schema myschema TO jack;
GRANT
```

- Revoke the usage permission for **myschema** from **jack**.

```
gaussdb=# REVOKE USAGE ON schema myschema FROM jack;
REVOKE
```

- Delete a schema.

- If a schema is empty, that is, it contains no database objects, you can run the **DROP SCHEMA** command. For example, drop an empty schema named **nullschema**.

```
gaussdb=# DROP SCHEMA IF EXISTS nullschema;
DROP SCHEMA
```

- To drop a schema that is not null, use the keyword **CASCADE** to drop it and all its objects. For example, drop **myschema** and all its objects.

```
gaussdb=# DROP SCHEMA myschema CASCADE;
DROP SCHEMA
```

- Drop user **jack**.
gaussdb=# **DROP USER jack;**
DROP ROLE

3.2.6.2 Creating and Managing Partitioned Tables

Context

GaussDB supports range partitioned tables, list partitioned tables, and hash partitioned tables.

- Range partitioned table: Data within a certain range is mapped to each partition. The range is determined by the partition key specified when the partitioned table is created. This partitioning mode is most commonly used. The partition key is usually a date. For example, sales data is partitioned by month.
- List partitioned table: Key values contained in the data are stored in different partitions, and the data is mapped to each partition in sequence. The key values contained in the partitions are specified when the partitioned table is created.
- Hash partitioned table: Data is mapped to each partition based on the internal hash algorithm. The number of partitions is specified when the partitioned table is created.

A partitioned table has the following advantages over an ordinary table:

- High query performance: You can specify partitions when querying partitioned tables, improving query efficiency.
- High availability: If a certain partition in a partitioned table is faulty, data in the other partitions is still available.
- Easy maintenance: To fix a partitioned table having a faulty partition, you only need to fix the partition.
- Balanced I/O: Partitions can be mapped to different disks to balance I/O and improve the overall system performance.

To convert an ordinary table to a partitioned table, you need to create a partitioned table and import data to it from the ordinary table. When you design tables, plan whether to use partitioned tables based on service requirements.

Procedure

This section uses range partitioning as an example to describe the related operations. For details about other partitioning examples, see [CREATE TABLE PARTITION](#). Before performing the operations in this section, ensure that you have created the **tpcds** schema.

Example 1: Use the default tablespace.

- Create a partitioned table (assuming that the **tpcds** schema has been created).

```
gaussdb=# CREATE TABLE tpcds.customer_address
(
  ca_address_sk integer NOT NULL ,
  ca_address_id character(16) NOT NULL ,
  ca_street_number character(10) ,
)
```

```

ca_street_name character varying(60) ,
ca_street_type character(15) ,
ca_suite_number character(10) ,
ca_city character varying(60) ,
ca_county character varying(30) ,
ca_state character(2) ,
ca_zip character(10) ,
ca_country character varying(20) ,
ca_gmt_offset numeric(5,2) ,
ca_location_type character(20)
)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE (ca_address_sk)
(
PARTITION P1 VALUES LESS THAN(5000),
PARTITION P2 VALUES LESS THAN(10000),
PARTITION P3 VALUES LESS THAN(15000),
PARTITION P4 VALUES LESS THAN(20000),
PARTITION P5 VALUES LESS THAN(25000),
PARTITION P6 VALUES LESS THAN(30000),
PARTITION P7 VALUES LESS THAN(40000),
PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;

```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

- Insert data.

Insert data from the **tpcds.customer_address** table to the **tpcds.web_returns_p2** table.

Suppose that the backup table **tpcds.web_returns_p2** of the **tpcds.customer_address** table has been created in the database. You can insert the data of the **tpcds.customer_address** table into the backup table **tpcds.web_returns_p2**.

```

gaussdb=# CREATE TABLE tpcds.web_returns_p2
(
ca_address_sk integer NOT NULL ,
ca_address_id character(16) NOT NULL ,
ca_street_number character(10) ,
ca_street_name character varying(60) ,
ca_street_type character(15) ,
ca_suite_number character(10) ,
ca_city character varying(60) ,
ca_county character varying(30) ,
ca_state character(2) ,
ca_zip character(10) ,
ca_country character varying(20) ,
ca_gmt_offset numeric(5,2) ,
ca_location_type character(20)
)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE (ca_address_sk)
(
PARTITION P1 VALUES LESS THAN(5000),
PARTITION P2 VALUES LESS THAN(10000),
PARTITION P3 VALUES LESS THAN(15000),
PARTITION P4 VALUES LESS THAN(20000),
PARTITION P5 VALUES LESS THAN(25000),
PARTITION P6 VALUES LESS THAN(30000),
PARTITION P7 VALUES LESS THAN(40000),
PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0

```

- Modify the row movement attributes of the partitioned table.
gaussdb=# ALTER TABLE *tpcds.web_returns_p2* DISABLE ROW MOVEMENT;
ALTER TABLE
- Delete a partition.
Delete partition **P8**.
gaussdb=# ALTER TABLE *tpcds.web_returns_p2* DROP PARTITION *P8*;
ALTER TABLE
- Add a partition.
Add partition **P8** and set its range to [40000,MAXVALUE).
gaussdb=# ALTER TABLE *tpcds.web_returns_p2* ADD PARTITION *P8* VALUES LESS THAN
(MAXVALUE);
ALTER TABLE
- Rename partitions.
 - Rename partition **P8** to **P_9**.
gaussdb=# ALTER TABLE *tpcds.web_returns_p2* RENAME PARTITION *P8* TO *P_9*;
ALTER TABLE
 - Rename partition **P_9** to **P8**.
gaussdb=# ALTER TABLE *tpcds.web_returns_p2* RENAME PARTITION FOR (40000) TO *P8*;
ALTER TABLE
- Query a partition.
Query partition **P6**.
gaussdb=# SELECT * FROM *tpcds.web_returns_p2* PARTITION (P6);
gaussdb=# SELECT * FROM *tpcds.web_returns_p2* PARTITION FOR (35888);
- Delete a partitioned table and its tablespaces.
gaussdb=# DROP TABLE *tpcds.customer_address*;
DROP TABLE
gaussdb=# DROP TABLE *tpcds.web_returns_p2*;
DROP TABLE

Example 2: Use a user-defined tablespace (assuming that the **tpcds** schema has been created).

Perform the following operations on the range partitioned table (the **tpcds** namespace in the example must be created in advance):

- Create tablespaces.
gaussdb=# CREATE TABLESPACE *example1* RELATIVE LOCATION 'tablespace1/tablespace_1';
gaussdb=# CREATE TABLESPACE *example2* RELATIVE LOCATION 'tablespace2/tablespace_2';
gaussdb=# CREATE TABLESPACE *example3* RELATIVE LOCATION 'tablespace3/tablespace_3';
gaussdb=# CREATE TABLESPACE *example4* RELATIVE LOCATION 'tablespace4/tablespace_4';

If the following information is displayed, the creation is successful:

```
CREATE TABLESPACE
```

- Create a partitioned table.
gaussdb=# CREATE TABLE *tpcds.customer_address*
(
 ca_address_sk integer NOT NULL ,
 ca_address_id character(16) NOT NULL ,
 ca_street_number character(10) ,
 ca_street_name character varying(60) ,
 ca_street_type character(15) ,
 ca_suite_number character(10) ,
 ca_city character varying(60) ,
 ca_county character varying(30) ,
 ca_state character(2) ,
 ca_zip character(10) ,
 ca_country character varying(20) ,
 ca_gmt_offset numeric(5,2) ,
 ca_location_type character(20)
)

```

TABLESPACE example1
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE (ca_address_sk)
(
  PARTITION P1 VALUES LESS THAN(5000),
  PARTITION P2 VALUES LESS THAN( 10000),
  PARTITION P3 VALUES LESS THAN( 15000),
  PARTITION P4 VALUES LESS THAN(20000),
  PARTITION P5 VALUES LESS THAN(25000),
  PARTITION P6 VALUES LESS THAN(30000),
  PARTITION P7 VALUES LESS THAN(40000),
  PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;

```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

- Insert data.

Insert data from the **tpcds.customer_address** table to the **tpcds.web_returns_p2** table.

Suppose that the backup table **tpcds.web_returns_p2** of the **tpcds.customer_address** table has been created in the database. You can insert the data of the **tpcds.customer_address** table into the backup table **tpcds.web_returns_p2**.

```

gaussdb=# CREATE TABLE tpcds.web_returns_p2
(
  ca_address_sk      integer          NOT NULL ,
  ca_address_id     character(16)    NOT NULL ,
  ca_street_number  character(10)    ,
  ca_street_name    character varying(60) ,
  ca_street_type    character(15)    ,
  ca_suite_number   character(10)    ,
  ca_city           character varying(60) ,
  ca_county         character varying(30) ,
  ca_state          character(2)      ,
  ca_zip           character(10)      ,
  ca_country        character varying(20) ,
  ca_gmt_offset     numeric(5,2)     ,
  ca_location_type  character(20)
)

```

```

TABLESPACE example1
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE (ca_address_sk)
(
  PARTITION P1 VALUES LESS THAN(5000),
  PARTITION P2 VALUES LESS THAN( 10000),
  PARTITION P3 VALUES LESS THAN( 15000),
  PARTITION P4 VALUES LESS THAN(20000),
  PARTITION P5 VALUES LESS THAN(25000),
  PARTITION P6 VALUES LESS THAN(30000),
  PARTITION P7 VALUES LESS THAN(40000),
  PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;

```

```
CREATE TABLE
```

```
gaussdb=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

- Modify the row movement attributes of the partitioned table.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
ALTER TABLE
```

- Delete a partition.

Delete partition **P8**.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE
```

- Add a partition.

Add partition **P8** and set its range to [40000,MAXVALUE).

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN  
(MAXVALUE);  
ALTER TABLE
```

- Rename partitions.

- Rename partition **P8** to **P_9**.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;  
ALTER TABLE
```

- Rename partition **P_9** to **P8**.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;  
ALTER TABLE
```

- Change the tablespaces of partitions.

- Change the tablespace of partition **P6** to **example3**.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P6 TABLESPACE example3;  
ALTER TABLE
```

- Change the tablespace of partition **P4** to **example4**.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P4 TABLESPACE example4;  
ALTER TABLE
```

- Query a partition.

Query partition **P6**.

```
gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P6);  
gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
```

- Delete a partitioned table and its tablespaces.

```
gaussdb=# DROP TABLE tpcds.customer_address;  
DROP TABLE  
gaussdb=# DROP TABLE tpcds.web_returns_p2;  
DROP TABLE  
gaussdb=# DROP TABLESPACE example1;  
gaussdb=# DROP TABLESPACE example2;  
gaussdb=# DROP TABLESPACE example3;  
gaussdb=# DROP TABLESPACE example4;  
DROP TABLESPACE
```

3.2.6.3 Creating and Managing Indexes

Context

Indexes accelerate data access but increase the processing time of insertion, update, and deletion operations. Therefore, before creating an index, consider whether it is necessary and select the columns where indexes are to be created. You can determine whether to create an index for a table by analyzing the service processing and data use of applications, as well as columns that are frequently used as search criteria or need to be collated.

Indexes are created based on columns in database tables. Therefore, you must correctly identify which columns require indexes. You are advised to create indexes for any of the following columns:

- Columns that are often searched and queried. This speeds up searches.
- Columns that function as primary keys. This enforces the uniqueness of the columns and the data collation structures in organized tables.
- Columns that are often joined. This increases the join efficiency.
- Columns that are often searched by range. The index helps collate data, and therefore the specified ranges are contiguous.

- Columns that often need to be collated. The index helps collate data, reducing the time for a collation query.
- Columns where the WHERE clause is executed frequently. This speeds up condition judgment.
- Columns that often appear after the keywords ORDER BY, GROUP BY, and DISTINCT.

NOTE

- After an index is created, the system automatically determines when to reference it. If the system determines that indexing is faster than sequential scan, the index will be used.
- After an index is successfully created, it must be synchronized with the associated table to ensure new data can be accurately located, which increases the data operation load. Therefore, delete unnecessary indexes periodically.
- Partitioned table indexes are classified into local indexes and global indexes. A local index corresponds to a specific partition, and a global index corresponds to the entire partitioned table.
- When logical replication is enabled, if you need to create a primary key index that contains system columns, you must set the **REPLICA IDENTITY** attribute of the table to **FULL** or use **USING INDEX** to specify a unique, non-local, non-deferrable index that does not contain system columns and contains only columns marked **NOT NULL**.

Procedure

To create a partitioned table, see [Creating and Managing Partitioned Tables](#).

- Create an index.
 - Create the partitioned table index **tpcds_web_returns_p2_index1** without specifying the partition name.

```
gaussdb=# CREATE INDEX tpcds_web_returns_p2_index1 ON tpcds.web_returns_p2 (ca_address_id) LOCAL;
```

If the following information is displayed, the creation is successful:

```
CREATE INDEX
```
 - Create the partitioned table index **tpcds_web_returns_p2_index2** with the partition name specified.

```
gaussdb=# CREATE INDEX tpcds_web_returns_p2_index2 ON tpcds.web_returns_p2 (ca_address_sk) LOCAL
(
  PARTITION web_returns_p2_P1_index,
  PARTITION web_returns_p2_P2_index TABLESPACE example3,
  PARTITION web_returns_p2_P3_index TABLESPACE example4,
  PARTITION web_returns_p2_P4_index,
  PARTITION web_returns_p2_P5_index,
  PARTITION web_returns_p2_P6_index,
  PARTITION web_returns_p2_P7_index,
  PARTITION web_returns_p2_P8_index
) TABLESPACE example2;
```

If the following information is displayed, the creation is successful:

```
CREATE INDEX
```
- Change the tablespace of an index partition.
 - Change the tablespace of index partition **web_returns_p2_P2_index** to **example1**.

```
gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION web_returns_p2_P2_index TABLESPACE example1;
```

If the following information is displayed, the modification is successful:

```
ALTER INDEX
```

- Change the tablespace of index partition **web_returns_p2_P3_index** to **example2**.

```
gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION
web_returns_p2_P3_index TABLESPACE example2;
```

If the following information is displayed, the modification is successful:

```
ALTER INDEX
```

- Rename an index partition.

Rename the index partition **web_returns_p2_P8_index** to **web_returns_p2_P8_index_new**.

```
gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 RENAME PARTITION
web_returns_p2_P8_index TO web_returns_p2_P8_index_new;
```

If the following information is displayed, the rename operation is successful:

```
ALTER INDEX
```

- Query indexes.

- Query all indexes defined by the system and users.

```
gaussdb=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
```

- Query information about a specified index.

```
gaussdb=# \di+ tpcds.tpcds_web_returns_p2_index2
```

- Delete indexes.

```
gaussdb=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
```

```
gaussdb=# DROP INDEX tpcds.tpcds_web_returns_p2_index2;
```

If the following information is displayed, the deletion is successful:

```
DROP INDEX
```

GaussDB supports four methods for creating indexes. For details, see [Table 3-4](#).

NOTE

- After an index is created, the system automatically determines when to reference it. If the system determines that indexing is faster than sequential scan, the index will be used.
- After an index is successfully created, it must be synchronized with the associated table to ensure new data can be accurately located, which increases the data operation load. Therefore, delete unnecessary indexes periodically.

Table 3-4 Indexing method

Indexing Method	Description
Unique index	An index that requires the uniqueness of an index attribute or an attribute group. If a table declares unique constraints or primary keys, GaussDB automatically creates unique indexes (or composite indexes) for columns that form the primary keys or unique constraints. Currently, unique indexes can be created only for the B-tree and UB-tree in GaussDB.
Composite index	An index that can be defined for multiple attributes of a table. Currently, composite indexes can be created only for B-tree in GaussDB and up to 32 columns can share a composite index.

Indexing Method	Description
Partial index	An index that can be created for subsets of a table. This indexing method contains only tuples that meet condition expressions.
Expression index	An index that is built on a function or expression calculated based on one or more attributes of a table. An expression index works only when the queried expression is the same as the created expression.

- Create an ordinary table.

```
gaussdb=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;
INSERT 0 0
```

- Create an ordinary index.

For the **tpcds.customer_address_bak** table, you need to perform the following operations frequently:

```
gaussdb=# SELECT ca_address_sk FROM tpcds.customer_address_bak WHERE ca_address_sk=14888;
```

Generally, the database system needs to scan the

tpcds.customer_address_bak table row by row to find all matched tuples. If the size of the **tpcds.customer_address_bak** table is large but only a few (possibly zero or one) of the WHERE conditions are met, the performance of this sequential scan is low. If the database system uses an index to maintain the **ca_address_sk** attribute, the database system only needs to search a few tree layers for the matched tuples. This greatly improves data query performance. Furthermore, indexes can improve the update and deletion operation performance in the database.

Create an index.

```
gaussdb=# CREATE INDEX index_wr_returned_date_sk ON tpcds.customer_address_bak
(ca_address_sk);
CREATE INDEX
```

- Create a unique index.

Create a unique index on the **SM_SHIP_MODE_SK** column in the **tpcds.ship_mode_t1** table.

```
gaussdb=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK);
```

- Create a composite index.

Frequently query records with **ca_address_sk** being **5050** and **ca_street_number** smaller than **1000** in the **tpcds.customer_address_bak** table.

```
gaussdb=# SELECT ca_address_sk,ca_address_id FROM tpcds.customer_address_bak WHERE
ca_address_sk = 5050 AND ca_street_number < 1000;
```

Define a composite index on the **ca_address_sk** and **ca_street_number** columns.

```
gaussdb=# CREATE INDEX more_column_index ON
tpcds.customer_address_bak(ca_address_sk ,ca_street_number );
CREATE INDEX
```

- Create a partial index.

If you only want to find records with **ca_address_sk** being **5050**, you can create a partial index to facilitate your query.

```
gaussdb=# CREATE INDEX part_index ON tpcds.customer_address_bak(ca_address_sk) WHERE
ca_address_sk = 5050;
CREATE INDEX
```

- Create an expression index.

Frequently query records with **ca_street_number** smaller than **1000**.

```
gaussdb=# SELECT * FROM tpcds.customer_address_bak WHERE trunc(ca_street_number) < 1000;
```

The following expression index can be created for this query task:

```
gaussdb=# CREATE INDEX para_index ON tpcds.customer_address_bak (trunc(ca_street_number));
CREATE INDEX
```

- Delete the **tpcds.customer_address_bak** table.

```
gaussdb=# DROP TABLE tpcds.customer_address_bak;
DROP TABLE
```

3.2.6.4 Creating and Managing Views

Context

If some columns in one or more tables in a database are frequently searched for, an administrator can define a view for these columns, and then users can directly access these columns in the view without entering search criteria.

Different from a base table, a view is a virtual table. Only view definition is stored in the database and view data is not. The data is stored in a base table. If data in the base table changes, the data in the view changes accordingly. A view is like a window through which users can know their interested data and data changes in the database. A view is triggered every time it is referenced.

Managing Views

- Create a view.

Create the **MyView** view. In the command, **tpcds.web_returns** indicates the created user table that contains the **wr_refunded_cash** integer column.

```
gaussdb=# CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.web_returns WHERE
trunc(wr_refunded_cash) > 10000;
CREATE VIEW
```

NOTE

The **OR REPLACE** parameter in this command is optional. It indicates that if the view exists, the new view will replace the existing view.

- Query a view.

Query **MyView**.

```
gaussdb=# SELECT * FROM MyView;
```

- Query views of the current user.

```
gaussdb=# SELECT * FROM my_views;
```

- Query all views.

```
gaussdb=# SELECT * FROM adm_views;
```

- Query details about a specified view.

View details about **MyView**.

```
gaussdb=# \d+ MyView
          View "PG_CATALOG.MyView"
  Column |      Type      | Modifiers | Storage | Description
-----+-----+-----+-----+-----
 USERNAME | CHARACTER VARYING(64) |          |          | extended |
View definition:
```

```
SELECT PG_AUTHID.ROLNAME::CHARACTER VARYING(64) AS USERNAME  
FROM PG_AUTHID;
```

- Delete a view.

Delete **MyView**.

```
gaussdb=# DROP VIEW MyView;  
DROP VIEW
```

3.2.6.5 Creating and Managing Sequences

Context

A sequence is a database object that generates unique integers. Sequence numbers are generated according to a certain rule. Sequences are unique because they increase automatically. This is why they are often used as primary keys.

You can create a sequence for a column in either of the following methods:

- Set the data type of a column to **sequence integer**. A sequence will be automatically created by the database for this column.
- Use the **CREATE SEQUENCE** statement to create a sequence. Set the initial value of the `nextval('sequence_name')` function to the default value of a column.

Procedure

Method 1: Set the data type of a column to a sequence integer. Example:

```
gaussdb=# CREATE TABLE T1  
(  
  id serial,  
  name text  
);
```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

Method 2: Create a sequence and set the initial value of the `nextval('sequence_name')` function to the default value of a column. You can cache a specific number of sequence values to reduce the requests to the GTM, improving the performance.

1. Create a sequence.

```
gaussdb=# CREATE SEQUENCE seq1 cache 100;
```

If the following information is displayed, the creation is successful:

```
CREATE SEQUENCE
```

2. Set the default value of a column so that the column has a unique identification attribute.

```
gaussdb=# CREATE TABLE T2  
(  
  id int not null default nextval('seq1'),  
  name text  
);
```

If the following information is displayed, the default value has been specified:

```
CREATE TABLE
```

3. Associate a sequence with a column.

Associate a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to.

```
gaussdb=# ALTER SEQUENCE seq1 OWNED BY T2.id;
```

If the following information is displayed, the operation is successful:

```
ALTER SEQUENCE
```

NOTE

The preceding methods are similar, except that the second method specifies cache for the sequence. A sequence having cache defined has inconsecutive values (such as 1, 4, and 5) and cannot maintain the order of its values. After the dependent column of a sequence has been specified, once the sequence is deleted, the sequence of the dependent will be deleted. A sequence shared by multiple columns is not forbidden in a database, but you are advised not to do that.

In the current version, you can specify the auto-increment column or set the default value of a column to **nextval('seqname')** when defining a table. You cannot add an auto-increment column or a column whose default value is **nextval('seqname')** to an existing table.

Precautions

Sequence values are generated by the GTM. By default, each request for a sequence value is sent to the GTM. The GTM calculates the result of the current value plus the step and then returns the result. The GTM is a globally unique node and is the performance bottleneck. Therefore, you are advised not to generate sequence values frequently and numerously, such as to use BulkLoad to import data. For example, the INSERT INTO SELECT FROM statement has poor performance in the following scenario:

```
gaussdb=# CREATE SEQUENCE newSeq1;
gaussdb=# CREATE TABLE newT1
(
    id int not null default nextval('newSeq1'),
    name text
);
gaussdb=# INSERT INTO newT1(name) SELECT name FROM T1;
```

Assume that data imported from table **T1** to table **newT1** has 10,000 rows. The following statements achieve better performance:

```
gaussdb=# INSERT INTO newT1(id, name) SELECT id,name FROM T1;
gaussdb=# SELECT SETVAL('newSeq1',10000);
```

NOTE

Rollback is not supported by sequence functions, including nextval() and setval(). The value of the setval function immediately takes effect on nextval in the current session in any cases and take effect in other sessions only when no cache is specified for them. If cache is specified for a session, it takes effect only after all the cached values have been used. To avoid duplicate values, use setval only when necessary. Do not set it to an existing sequence value or a cached sequence value.

To generate the default sequence value using BulkLoad, set sufficient cache for **newSeq1** and do not set **Maxvalue** or **Minvalue**. The database will push down the calling of **nextval('sequence_name')** to DNs to improve performance. Currently, the concurrent connection requests that can be processed by the GTM are limited. If there are too many DNs, a large number of concurrent connection requests will be sent to the GTM. In this case, you need to limit the bulk loading

concurrency, so that DNs do not fully occupy GTM connections. If the target table is a duplicate table (DISTRIBUTE BY REPLICATION), pushdown cannot be performed. When the data volume is large, the performance is affected and the space may bloat sharply. After the import is complete, you need to run the **vacuum full** command to restore the data. The best way is not to use BulkLoad to generate the default sequence value.

After a sequence is created, one single-row table is maintained on each node to store the sequence definition and value, which is obtained from the last interaction with the GTM rather than updated in real time. The single-row table on a node does not update when other nodes request a new value from the GTM or when the sequence is modified using setval.

3.2.6.6 Creating and Managing Scheduled Jobs

Context

Time-consuming jobs, such as summarizing statistics or synchronizing data from another database, affect service performance if they are performed during the daytime and incur overtime hours if performed at night. To solve this problem, the database is compatible with the scheduled job function in the ORA database. You can create scheduled jobs that are automatically triggered to reduce O&M workload.

This function calls APIs provided by the **DBE_SCHEDULER** and **DBE_TASK** packages to create scheduled jobs, execute jobs automatically, delete jobs, and modify job attributes (including job ID, the enabled/disabled status of a job, job triggering time, triggering interval, and job contents). The DBE_SCHEDULER API is recommended to ensure high availability and reliability and support more flexible job scheduling. For details about the API description and migration guide examples, see [DBE_SCHEDULER](#).

Managing Scheduled Jobs

Step 1 Create a test table.

```
gaussdb=# CREATE TABLE test(id int, time date);
```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

Step 2 Create a user-defined stored procedure.

```
gaussdb=# CREATE OR REPLACE PROCEDURE PRC_JOB_1()  
AS  
N_NUM integer :=1;  
BEGIN  
FOR I IN 1..1000 LOOP  
INSERT INTO test VALUES(I,SYSDATE);  
END LOOP;  
END;  
/
```

If the following information is displayed, the creation is successful:

```
CREATE PROCEDURE
```

Step 3 Create a job.

- Create a job with unspecified **job_id** and execute the **PRC_JOB_1** stored procedure every minute.

```
gaussdb=# call db_task.submit('call public.prc_job_1(); ', sysdate, 'interval "1 minute"', :a);
job
-----
1
(1 row)
```

- Specify **job_id** to create a job. The value of **job_id** ranges from 1 to 32767.

```
gaussdb=# call db_task.id_submit(2,'call public.prc_job_1(); ', sysdate, 'interval "1 minute"');
isubmit
-----
(1 row)
```

Step 4 View details of jobs created by the current user.

```
gaussdb=# select job,dbname,start_date,last_date,this_date,next_date,broken,status,interval,failures,what
from my_jobs;
job | dbname | start_date | last_date | this_date | next_date | broken | status | interval | failures | what
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
1 | testdb | 2017-07-18 11:38:03 | 2017-07-18 13:53:03.607838 | 2017-07-18 13:53:03.607838 | 2017-07-18 13:54:03 | n | s | interval '1 minute' | 0 | call public.prc_job_1();
(1 row)
```

Step 5 Stop a job.

```
gaussdb=# call db_task.finish(1,true);
broken
-----
(1 row)
```

Step 6 Start a job.

```
gaussdb=# call db_task.finish(1,false);
broken
-----
(1 row)
```

Step 7 Modify job attributes.

- Modify the **next_time** parameter information about the job.

```
-- Set next_time of Job1 to 1 hour so that Job1 will be executed in one hour.
gaussdb=# call db_task.next_time(1, sysdate+1.0/24);
next_date
-----
(1 row)
```

- Modify the **Interval** parameter about a job.

```
-- Set Interval of Job1 to 1 hour so that Job1 will be executed every one hour.
gaussdb=# call db_task.interval(1,'sysdate + 1.0/24');
interval
-----
(1 row)
```

- Modify the **What** parameter about a job.

```
-- Set What to the SQL statement insert into public.test values(333, sysdate+5); for Job1.
gaussdb=# call db_task.content(1,'insert into public.test values(333, sysdate+5);');
what
-----
(1 row)
```

- Modify **Next_date**, **Interval**, and **What** parameters about a job.

```
gaussdb=# call dbe_task.update(1, 'call public.prc_job_1()'; sysdate, 'interval "1 minute");
change
-----
(1 row)
```

Step 8 Remove a job.

```
gaussdb=# call dbe_task.cancel(1);
remove
-----
(1 row)
```

Step 9 View the job execution status.

If a job fails to be automatically executed (that is, the value of **job_status** is 'f'), contact the administrator to view the pg_log run log to view the failure information of the job.

From **detail error msg**, you can see the failure causes.

```
LOG: Execute Job Detail:
job_id: 1
what: call public.test();
start_date: 2017-07-19 23:30:47.401818
job_status: failed
detail error msg: relation "test" does not exist
end_date: 2017-07-19 23:30:47.401818
next_run_date: 2017-07-19 23:30:56.855827
```

Step 10 Set job permissions.

- During the creation of a job, the job is bound to the user and database that created the job. Accordingly, the user and database are added to **dbname** and **log_user** columns in the pg_job system catalog, respectively.
- If the current user is a database administrator, system administrator, or the user (**log_user** in pg_job) who created the job, the user has permissions to delete or modify job parameters using the **remove**, **change**, **next_data**, **what**, or **interval** parameter. Otherwise, the system displays a message indicating that the user has no permissions to perform operations on this job.
- If the current database is the one that created a job (that is, **dbname** in pg_job), you can delete or modify parameter settings of the job using the **cancel**, **update**, **next_data**, **content**, or **interval** parameter.
- When deleting the database that created a job (that is, **dbname** in pg_job), the system automatically deletes the job records of the database.
- When deleting the user who created a job (that is, **log_user** in pg_job), the system automatically deletes the job records of the user.

Step 11 Manage job concurrency.

You can configure the GUC parameter **job_queue_processes** to adjust the number of jobs running at the same time.

- Setting **job_queue_processes** to **0** indicates that the scheduled job function is disabled and all jobs will not be executed.
- Setting **job_queue_processes** to a value that is greater than **0** indicates that the scheduled job function is enabled and this value is the maximum number of jobs that can be concurrently processed.

Too many concurrent jobs consume many system resources, so you need to set the number of concurrent jobs to be processed. If the current number of concurrent

jobs reaches the value of **job_queue_processes** and some of them expire, these jobs will be postponed to the next polling period. Therefore, you are advised to set the polling interval (the **Interval** parameter of the **submit** API) based on the execution duration of each job to avoid the problem that jobs in the next polling period cannot be properly processed because of overlong job execution time.

Note: For clusters that do not use jobs, set **job_queue_processes** to **0** to disable job functions to reduce the resource consumption.

----End

4 Development and Design Proposal

4.1 Overview

Overview

This chapter describes the database-related design and development specifications in the product design and development process based on the product life cycle.

This chapter aims to improve the readability and code quality, and emphasizes the practicability and operability. It specifies the issues that may occur during the database development. The specifications are as follows:

Design specifications in terms of database and performance

Programming specifications in terms of typesetting, naming, comments, syntax, scripts, database programming, installation and deployment, and security

In addition, detailed rules and specific examples are provided for some specifications if necessary.

The development specifications are for reference only. The specific specifications should be comprehensively evaluated and implemented by database users based on the application technical architecture and planning.

Terms

This document uses the following terms for description:

- **Specification:** database specification, which must be complied with during programming and design. Otherwise, the database reports an error.
- **Rule:** a convention that during programming and design.
- **Recommendation:** a convention that during programming and design.
- **Description:** explanation of the rule or recommendation in question.
- **Example:** a positive or negative example of the rule or recommendation.
- **Sharding:** Data in a table is split based on a specified policy and stored in tables with the same name on multiple DNs. The data stored in these tables does not overlap with each other, which can be considered as horizontal

sharding of tables, for example, the **t_user** table is split into four shards to four DNs by hashing the primary key **userId**. Each shard has a **t_user** table with the same name.

Applicability

This document is intended for designers, developers, development database administrators (DBAs), operation and maintenance (O&M) DBAs, and O&M personnel.

4.2 Database Design Specifications

4.2.1 General Specifications

- [Rule] Database valuable features

Table 4-1 Recommended database valuable features

Category	Feature List	Description
Table type	DISTRIBUTION table	Data sharding.
	PARTITION table	Data partitioning.
Storage engine	Row store	Stores tables by row, which is recommended in the point query and a scenario with many add, delete, and modify operations.
Transaction	Transaction block	Explicitly starts a transaction.
	Single-statement transaction	Does not explicitly start a transaction. A single statement is a transaction.
	Distributed transaction (weak consistency)	GTM-free mode. Strong consistency can be ensured in the sharding scenario.
Scale-out	Online scale-out	Ensures smooth transition of user services during node scale-out and data redistribution.
Deployment	One primary and multiple standby DNs	Supports DN sharding. One primary DN and multiple standby DNs are used to ensure data backup and disaster recovery.
Security	Transparent data encryption	Supports database-level storage encryption. Upper-layer services are unaware of the encryption.
Data type	Integer	TINYINT, SMALLINT, INTEGER, BIGINT
	Arbitrary-precision	NUMERIC/DECIMAL

Category	Feature List	Description
	Floating-point	REAL/FLOAT4,DOUBLE PRECISION/ FLOAT8,FLOAT
	Boolean	Boolean
	Fixed-length character	CHAR(n)
	Variable-length character	VARCHAR(n), NVARCHAR2(n), and TEXT
	Time	DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPZ, SMALLDATETIME, INTERVAL, and REALTIME
	Binary	BYTEA (variable-length binary)
	Bit string	BIT(n) and VARBIT(n)
Function	String function	String data type processing function.
	Binary string function	Binary string type processing function.
	Numeric operation function	Numeric type processing function.
	Time and date processing function	Time and date type processing function.
Index	Primary key or unique index	Single-column or multi-column primary key or unique index.
	B-tree index	Index type.

- [Rule] Recommended best practices for database usage

Table 4-2 Recommended best practices for database usage

No.	Item	Recommended Value
1	Optimal number of shards in a cluster (number of primary DN's)	< 256
2	Optimal number of long connections in a cluster	For details, see the default configuration of GUC parameter max_connections in the corresponding hardware specifications.

No.	Item	Recommended Value
3	Maximum data volume on a single physical node	16 TB (Determine the capacity based on the backup and restoration specifications.)
4	Number of active databases in a cluster	1
5	Total number of cluster tables and indexes	10000 (It is recommended that the number of tables in a single schema be less than or equal to 200.) NOTE The value is only a recommended value. You need to plan the value based on the disk capacity and customer services. Theoretically, the maximum value is 2^{32} .
6	Optimal number of columns in a single table	< 50
7	Optimal number of indexes in a table	< 5
8	Optimal number of composite indexes in a table	< 3
9	Optimal number of columns in a single composite index	< 5
10	Optimal width of a single row	< 2000
11	Optimal capacity of a single column	< 10 MB
12	Optimal SQL statement length	< 5000
13	Percentage of available disk space	50%

4.2.2 Deployment Specifications

[Rule] After a database is installed, DBA must configure the database GUC parameters properly based on the environment features.

 NOTE

For details about GUC parameter configuration, contact the administrator..

4.2.3 Database Object Naming Specifications

- [Specification] Do not use reserved or non-reserved keywords to name database objects. You can run the following command to query the database keyword, or view it in the [Keywords](#) section:

```
SELECT * FROM pg_get_keywords();
```
- [Specification] The database object name contains a maximum of 63 bytes.

Table 4-3 Restriction on the length of database object names

Object Type	Length Restriction (Unit: Byte)	Remarks
Database name	≤ 63	-
Table name	≤ 63	-
Table object	≤ 63	If the length of a table name exceeds this value, the kernel truncates the table name. As a result, the actual name is inconsistent with the configured value. In addition, characters may be truncated in different character sets and unexpected characters may appear.
Temporary table not used by services	≤ 63	For example, an intermediate table used by operation personnel for temporary backup or data collection. Naming rule: tmp_Table name abbreviation_Creator account abbreviation_Creation date , for example, tmp_user_ytw_160505 .
Table column	≤ 63	Meaningful English words or abbreviations are used to name the column. For example, a column of the bool type is named in the format of is_Description . A column whose member status is enabled in the member table is named is_enabled .
View name and column name	≤ 63	-
Function name and parameter name	≤ 63	-

- [Rule] Do not use a string enclosed in double quotation marks (") to define the database object name, unless you must specify its capitalization.
 - By default, GaussDB is case-insensitive for object names in SQL statements. If two different tables (**t_Table** and **t_table**) exist in the same database, use double quotation marks ("). Otherwise, avoid using double quotation marks (").
 - Case sensitivity of database object names makes fault locating difficult.
- [Rule] Identifiers must meet the following requirements:
 - An identifier starts with a letter or underscore (_) and can contain letters, digits, underscores (_), dollar signs (\$), and number signs (#).
 - If an identifier is enclosed in backquotes (`) in B-compatible mode or double quotation marks ("), any combination of valid characters can be used, for example, "123gs_column".
 - Identifiers are case-insensitive. They are case-sensitive only when they are enclosed in backquotes (`) in B-compatible mode or double quotation marks (").
- [Recommendation] Use the same naming style for database objects.
 - In a system undergoing incremental development or service migration, you are advised to comply with its historical naming conventions.
 - You are advised to use multiple words separated with underscores (_).
 - You are advised to use intelligible names and common acronyms or abbreviations for database objects. For example, you can use English words or Chinese pinyin indicating actual business terms. The naming format should be consistent within a cluster.
 - A variable name must be descriptive and meaningful. It must have a prefix indicating its type.
- [Recommendation] The name of a table object should indicate its main characteristics, for example, whether it is an ordinary, temporary, or unlogged table.
 - An ordinary table name should indicate the business relevant to a dataset.
 - Temporary tables are named in the format of **tmp_Suffix**.
 - Unlogged tables are named in the format of **ul_Suffix**.
 - Foreign tables are named in the format of **f_Suffix**.
 - Do not create database objects whose names start with **redis_**.
 - Do not create database objects named **pgxc_redistb**.
 - Do not create database objects whose names start with **mlog_** or **matviewmap_**.
 - Do not create database objects whose names start with **gs_role_**.

4.2.4 Database and Schema Design Specifications

- [Description] When you create a database, exercise caution when you set **ENCODING** and **DBCOMPATIBILITY** configuration items. GaussDB supports the TD-, ORA-, MYSQL-, PG-compatible, and M-compatible modes which are compatible with the Teradata syntax, Oracle syntax, MySQL syntax, PG syntax, and M-compatible syntax, respectively. The syntax behavior varies according to the compatibility mode. By default, the MySQL-compatible mode is used.

- [Description] By default, a database owner has all permissions for all objects in the database, including the deletion permission. Exercise caution when deleting a permission.
- [Description] To let a user access an object in a schema, assign the usage permission and the permissions for the object to the user, unless the user has the SYSADMIN permission or is the schema owner.
- [Description] To let a user create an object in the schema, grant the CREATE permission for the schema to the user.
- [Description] By default, a schema owner has all permissions for all objects in the schema, including the deletion permission. Exercise caution when deleting a permission.
- [Specification] The database name must be specified when the JDBC client is used to connect to the database. The format is as follows:

```
jdbc:postgresql://ip:port/database_name
```

NOTICE

Once a JDBC instance is created, the database cannot be switched.

- [Specification] The database does not support case-insensitive collation.
- [Rule] Before using services, the system administrator must create databases, schemas, and users for services, and then grant object permissions to corresponding users.
- [Rule] Create a service database before using a service. Do not use the Postgres database created by default after the database is installed to store service data. You are advised to create your own database based on service requirements.
- [Rule] When creating a database, you set the character set to **UTF8** and must select the character set that is the same as that of the client.

To meet globalization requirements, database encoding should store and identify most characters. Therefore, UTF8 is recommended. The UTF8 character set is equivalent to the UTF8MB4 character set in MySQL and supports emoji.

If the encoding mode of the client is different from that of the database, transcoding is required, affecting performance. In addition, kernel optimization for the same encoding cannot be triggered, affecting the query efficiency.

To change the character set of the client, perform the following steps:

- Set client connection parameters. Take JDBC as an example. Add **characterEncoding** and **allowEncodingChanges** to the URL.

```
jdbc:postgresql://ip:port/database_name?characterEncoding=utf8&allowEncodingChanges=true
```
- Modify the database GUC parameter.

```
SET client_encoding = 'UTF8';
```

Set the database encoding when executing the **CREATE DATABASE**.

```
CREATE DATABASE tester WITH ENCODING = 'UTF8';
```

NOTICE

The character set cannot be changed once the database is created.

- [Recommendation] In a cluster, the recommended number of user-defined databases is 3. It is recommended that the number of user-defined databases be less than or equal to 10. If there are too many user-defined databases, O&M operations, such as upgrade and backup, will be inefficient.
- [Recommendation] It is recommended that the number of schemas in the actual user environment be no more than 100. If there are too many schemas in a database, operations that depend on the number of schemas, such as `gs_dump`, becomes slow.
- [Recommendation] You are advised to use schemas to isolate services for convenience and resource sharing.

Both database and schema can be used to isolate services. Differences are as follows:

- Databases share little resources. Connections to and permissions on them are also isolated. However, the databases cannot access each other. The database must be specified during JDBC connection establishment. After the connection is established, the database cannot be switched.
- Schemas share more resources than databases do. User permissions on schemas and subordinate objects can be controlled using the GRANT and REVOKE syntax.
- [Recommendation] You are advised to specify `LC_CTYPE` and `LC_COLLATE` when creating a database. These parameters will affect the data collation sequence.

Example:

```
CREATE DATABASE SAMPLE_DB WITH LC_CTYPE = 'zh_CN.gbk' LC_COLLATE = 'zh_CN.gbk';
```

4.2.5 Permission Design Specifications

- [Rule] The initial database user can only connect and access a database for DBA management. Services are not allowed to directly log in as this user to connect to and access the database.

NOTE

Initial database user, that is, the database installation user, which has the same name as the OS user to which the cluster belongs.

- [Rule] The initial database user creates a database and users for services. Services use the created users to log in to and access the database.
- [Rule] Assign permissions to roles and users based on the least privilege principle.
- [Recommendation] Manage permissions by roles instead of users.

Use roles to manage permissions, that is, configure permissions for roles and grant the roles to users.

Roles facilitate permission management in scenarios such as multiple users and user changes. Example:

- Roles and users are in many-to-many relationship. A role can be granted to multiple users. If permissions of a role are modified, the permissions of the granted users can be updated at the same time.
- Deleting a user does not affect the role.
- A new user can quickly obtain required permissions by granting a role to the user.

- [Recommendation] Avoid the risk of permission exploitation due to improper use of permissions.

In some scenarios, improper user operations may cause permission exploitation. Example:

- When creating a non-system type by associating a function, the user needs to understand the definition of the type and the function associated with the type. If this function is not properly used, permissions may be exploited due to the associated function.
- When GRANT is used to grant a user the permission to use a table, if the permission is not properly used, ALTER may be used to add expressions to the default values and constraints of the table, or indexes may be created to add expressions to INDEX. In this case, the permission may be exploited.
- When GRANT is used to grant the TRIGGER permission, if the permission is not properly used, the WHEN condition may be used to create expressions. When the TRIGGER is triggered, the permission may be exploited.

4.2.6 Character Set Design Specifications

- [Specification] Currently, character sets can be defined only for databases.
- [Recommendation] The database and client use the same UTF-8 character set.
 - Clients include ODBC, JDBC, and GSQL.
 - For details about the reason why the UTF-8 character set is recommended, see [Database Design Specifications](#).

4.2.7 Table Design Specifications

- [Rule] View nesting is not recommended.

If wildcards are used during view compilation, an error occurs in the view when columns are added to or deleted from the called view. At the same time, view nesting may cause low execution efficiency because indexes cannot be used. Therefore, you are advised to use base tables with indexes instead of performing join operations on views.
- [Description] Evenly scan each DN when querying tables. Otherwise, DN's most frequently scanned will become the performance bottleneck. For example, when you use equivalent filter conditions on a fact table, the nodes are not evenly scanned.
- [Recommendation] Minimize random I/Os. Through clustering, you can sequentially store hot data, converting random I/O to sequential I/O to reduce the cost of I/O scanning.
- [Recommendation] Try to avoid data shuffling. To shuffle data is to physically transfer it from one node to another. This unnecessarily occupies many network resources. To reduce network pressure, locally process data, and improve cluster performance and concurrency, you can minimize data shuffling by using proper join and grouping conditions.
- [Recommendation] Try to avoid collation operations when defining a view. ORDER BY is invalid in the top-level view. If you must collate the output data, use ORDER BY in a called view.

Selecting a Distribution Mode

Table 4-4 shows how to select a table distribution mode.

Table 4-4 Table distribution modes and scenarios

Distribution Mode	Description	Application Scenario
Hash	Table data is distributed on all DNs in a cluster by hash.	Fact tables containing a large amount of data.
Replication	Full data in a table is stored on every DN in the cluster.	Dimension tables and fact tables containing a small amount of data.
Range	Table data is mapped to specified columns based on the range and distributed to the corresponding DNs.	Users need to customize distribution rules.
List	Table data is mapped to specified columns based on specific values and distributed to corresponding DNs.	Users need to customize distribution rules.

- [Description] Evenly distribute table data on each DN to prevent data skew. If most data is stored on several DNs, the effective capacity of a cluster decreases. Select a proper distribution key to avoid data skew.
- [Rule] DISTRIBUTE BY must be specified and the table distribution policy must comply with the following principles.

GaussDB currently offers four table distribution strategies: replication, hash, range, and list. REPLICATION retains a complete data table on each node. HASH distributes table data to multiple nodes based on the provided distribution key values. Range and list map data to corresponding target nodes based on the value of the distribution key falling within a specific range or specific values.

- If the number of data records in a system configuration table or data dictionary table is less than 20 million and the INSERT and UPDATE operations are of low frequency, REPLICATION distribution policy is required.

```
CREATE TABLE t1 (contentId INT) DISTRIBUTE BY REPLICATION;
```

NOTICE

Exercise caution when using the REPLICATION distribution. This distributed table may cause space expansion and DML performance deterioration.

- For tables that do not meet the preceding requirement, that is, tables with large data volumes and high update frequency, data must be sharded and the HASH distribution policy must be used. The distribution key must be one or more columns in the primary key.

```
CREATE TABLE t1 (contentId INT) DISTRIBUTE BY HASH (contentId);
```

- Users can use the range and list distribution policies to customize distribution rules.

```
CREATE TABLE t1 (contentId INT) DISTRIBUTE BY Range (contentId)
(
  SLICE s1 VALUES LESS THAN (10) DATANODE dn1,
  SLICE s2 VALUES LESS THAN (20) DATANODE dn2,
  SLICE s3 VALUES LESS THAN (30) DATANODE dn3,
  SLICE s4 VALUES LESS THAN (MAXVALUE) DATANODE dn4
);
```

```
CREATE TABLE t1 (contentId INT) DISTRIBUTE BY List (contentId)
(
  SLICE s1 VALUES (10) DATANODE dn1,
  SLICE s2 VALUES (20) DATANODE dn2,
  SLICE s3 VALUES (30) DATANODE dn3,
  SLICE s4 VALUES (DEFAULT) DATANODE dn4
);
```

- You are advised not to shard a table with less than 20 million data records, has multiple rows of data inserted and updated, and has a high frequency of range query. Currently, such tables are not supported but will be provided in the next version. Therefore, the hash-distributed table is used.

The example of defining a distributed table is as follows:

-- Define a table with each row stored in all DNs.

```
CREATE TABLE warehouse_d1
(
  W_WAREHOUSE_SK      INTEGER      NOT NULL,
  W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
  W_WAREHOUSE_NAME    VARCHAR(20)
  W_WAREHOUSE_SQ_FT   INTEGER
  W_STREET_NUMBER     CHAR(10)
  W_STREET_NAME       VARCHAR(60)
  W_STREET_TYPE       CHAR(15)
  W_SUITE_NUMBER      CHAR(10)
  W_CITY              VARCHAR(60)
  W_COUNTY            VARCHAR(30)
  W_STATE             CHAR(2)
  W_ZIP               CHAR(10)
  W_COUNTRY           VARCHAR(20)
  W_GMT_OFFSET        DECIMAL(5,2)
)DISTRIBUTE BY REPLICATION;
```

-- Define a hash table.

```
CREATE TABLE warehouse_d2
(
  W_WAREHOUSE_SK      INTEGER      NOT NULL,
  W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
  W_WAREHOUSE_NAME    VARCHAR(20)
  W_WAREHOUSE_SQ_FT   INTEGER
  W_STREET_NUMBER     CHAR(10)
  W_STREET_NAME       VARCHAR(60)
  W_STREET_TYPE       CHAR(15)
  W_SUITE_NUMBER      CHAR(10)
  W_CITY              VARCHAR(60)
  W_COUNTY            VARCHAR(30)
  W_STATE             CHAR(2)
  W_ZIP               CHAR(10)
  W_COUNTRY           VARCHAR(20)
  W_GMT_OFFSET        DECIMAL(5,2),
  CONSTRAINT W_CONSTR_KEY3 UNIQUE(W_WAREHOUSE_SK)
```

```
)DISTRIBUTE BY HASH(W_WAREHOUSE_SK);
-- Define a table using range distribution.
CREATE TABLE warehouse_d3
(
W_WAREHOUSE_SK      INTEGER      NOT NULL,
W_WAREHOUSE_ID      CHAR(16)     NOT NULL,
W_WAREHOUSE_NAME    VARCHAR(20)
W_WAREHOUSE_SQ_FT   INTEGER
W_STREET_NUMBER     CHAR(10)
W_STREET_NAME       VARCHAR(60)
W_STREET_TYPE       CHAR(15)
W_SUITE_NUMBER      CHAR(10)
W_CITY              VARCHAR(60)
W_COUNTY            VARCHAR(30)
W_STATE             CHAR(2)
W_ZIP              CHAR(10)
W_COUNTRY           VARCHAR(20)
W_GMT_OFFSET        DECIMAL(5,2)
)DISTRIBUTE BY RANGE(W_WAREHOUSE_ID)
(
SLICE s1 VALUES LESS THAN (10) DATANODE dn1,
SLICE s2 VALUES LESS THAN (20) DATANODE dn2,
SLICE s3 VALUES LESS THAN (30) DATANODE dn3,
SLICE s4 VALUES LESS THAN (MAXVALUE) DATANODE dn4
);
-- Define a table using list distribution.
CREATE TABLE warehouse_d4
(
W_WAREHOUSE_SK      INTEGER      NOT NULL,
W_WAREHOUSE_ID      CHAR(16)     NOT NULL,
W_WAREHOUSE_NAME    VARCHAR(20)
W_WAREHOUSE_SQ_FT   INTEGER
W_STREET_NUMBER     CHAR(10)
W_STREET_NAME       VARCHAR(60)
W_STREET_TYPE       CHAR(15)
W_SUITE_NUMBER      CHAR(10)
W_CITY              VARCHAR(60)
W_COUNTY            VARCHAR(30)
W_STATE             CHAR(2)
W_ZIP              CHAR(10)
W_COUNTRY           VARCHAR(20)
W_GMT_OFFSET        DECIMAL(5,2)
)DISTRIBUTE BY LIST(W_COUNTRY)
(
SLICE s1 VALUES ('USA') DATANODE dn1,
SLICE s2 VALUES ('CANADA') DATANODE dn2,
SLICE s3 VALUES ('UK') DATANODE dn3,
SLICE s4 VALUES (DEFAULT) DATANODE dn4
);
```

For details about the table distribution syntax, see [CREATE TABLE](#).

Selecting a Distribution Key

A distribution key is important for a distributed table. An improper distribution key may cause data skew. As a result, the I/O load is heavy on several DNs, affecting the overall query performance. Therefore, after determining the distribution policy of a distributed table, you need to check the table data skew to ensure that data is evenly distributed. Comply with the following rules to select a distribution key:

- [Recommendation] Select a column containing discrete data as the distribution key, so that data can be evenly distributed on each DN. If the data in a single column is not discrete enough, consider using multiple columns as distribution keys. You can select the primary key of a table as the

distribution key. For example, in an employee information table, select the certificate number column as the distribution key.

- [Recommendation] If the first rule is met, do not select a column having constant filter conditions as the distribution key. For example, in a query on the **dwcjk** table, if the **zqdh** column contains the constant filter condition **zqdh='000001'**, avoid selecting the **zqdh** column as the distribution key.
- [Recommendation] If the first and second rules are met, select the join conditions in a query as distribution keys. If a join condition is used as a distribution key, the data involved in a join task is locally distributed on DNs, which greatly reduces the data flow cost among DNs.
- [Recommendation] It is recommended that the number of distribution key columns be less than or equal to 3. Too many columns will cause high computing overhead.
- [Rule] Properly design distribution keys to facilitate query development and ensure even data storage, avoiding data skew and read hotspots.
 - a. Use columns with discrete values as distribution keys so that data can be evenly distributed to each DN.
 - b. If the preceding requirement is met, it is not recommended that columns with constant filtering be used as distribution keys. Otherwise, all query tasks will be distributed to a unique and fixed DN.
 - c. If the preceding two requirements are met, select the JOIN condition as the distribution key. In this way, related data of the JOIN task is distributed on the same DN, reducing the cost of data flow between DNs.

Table 4-5 Common distribution keys and effects

Distribution Key Value	Distribution Effect
User ID. Many users exist in an application.	Good
Status code. Only several status codes are available.	Poor
Date when a project is created. The value is rounded off to the latest time segment (for example, day, hour, or minute).	Poor
Device ID. Each device accesses data at a relatively similar interval.	Good
Device ID. Many devices are traced. However, one device is more commonly used than all other devices.	Poor

- [Rule] The length of the column used by the distribution key cannot exceed 128 characters. If the length is too long, the computing overhead is high.
- [Rule] Once a distribution key is inserted, it cannot be updated unless you delete the key and insert it again.

Selecting a Partitioning Mode

Table 4-6 Table partitioning modes and scenarios

Partitioning Mode	Description
Range	Table data is partitioned by range.
List	Table data is partitioned by a specified column based on a specific value.
Hash	Table data is partitioned by hash.

Comply with the following rules to partition a table containing a large amount of data:

- [Description] Reduce the amount of data to be scanned. You can use the pruning mechanism of a partitioned table.
- [Recommendation] Create partitions on columns that indicate certain ranges, such as dates and regions.
- [Recommendation] A partition name should show the data characteristics of a partition. For example, its format can be *Keyword+ Range* characteristics.
- [Recommendation] Define the upper boundary value of the partition as **MAXVALUE** to prevent data overflow.

The example of defining a partitioned table is as follows:

```
-- Create a range partitioned table.
CREATE TABLE staffs_p1
(
  staff_ID    NUMBER(6) not null,
  FIRST_NAME  VARCHAR2(20),
  LAST_NAME   VARCHAR2(25),
  EMAIL       VARCHAR2(25),
  PHONE_NUMBER VARCHAR2(20),
  HIRE_DATE   DATE,
  employment_ID VARCHAR2(10),
  SALARY      NUMBER(8,2),
  COMMISSION_PCT NUMBER(4,2),
  MANAGER_ID  NUMBER(6),
  section_ID  NUMBER(4)
)
PARTITION BY RANGE (HIRE_DATE)
(
  PARTITION HIRE_19950501 VALUES LESS THAN ('1995-05-01 00:00:00'),
  PARTITION HIRE_19950502 VALUES LESS THAN ('1995-05-02 00:00:00'),
  PARTITION HIRE_maxvalue VALUES LESS THAN (MAXVALUE)
);

-- Create a list partitioned table.
CREATE TABLE test_list (col1 int, col2 int)
partition by list(col1)
(
  partition p1 values (2000),
  partition p2 values (3000),
  partition p3 values (4000),
  partition p4 values (5000)
);

-- Create a hash partitioned table.
CREATE TABLE test_hash (col1 int, col2 int)
```

```
partition by hash(col1)
(
partition p1,
partition p2
);
```

For details about the table partition syntax, see [CREATE TABLE PARTITION](#).

4.2.8 Column Design Specifications

- [Rule] Use recommended data types for column design.
Recommended data types must be used for column design. Some data types are not recommended because they apply to limited service scenarios and are not used on a large scale for commercial purposes.

Table 4-7 Best practices of database data types

Data Type	Description	Recommended or Not
UUID	Different clusters may generate the same UUID.	Prohibited
Serial integer	Auto-increment column, including SMALLSERIAL, SERIAL, and BIGSERIAL.	Prohibited
Integer	TINYINT, SMALLINT, INTEGER, and BIGINT	Recommended
Arbitrary-precision	NUMERIC/DECIMAL	Recommended
Floating-point	REAL/FLOAT4, DOUBLE PRECISION/FLOAT8, and FLOAT	Recommended
Boolean	Boolean	Recommended
Fixed-length character	CHAR(<i>n</i>)	Recommended
Variable-length character	VARCHAR(<i>n</i>) and NVARCHAR2(<i>n</i>)	Recommended
	TEXT and CLOB (character large object)	Not recommended
Time	DATE, TIME, TIMESTAMP, SMALLDATETIME, INTERVAL, and REALTIME	Recommended
	TIMETZ and TIMESTAMPTZ	Not recommended
Binary	BYTEA (variable-length binary)	Recommended

Data Type	Description	Recommended or Not
	BLOB (binary large object) and RAW (variable-length hexadecimal string)	Not recommended
Bit string	BIT(n) and VARBIT(n)	Recommended
Special character	NAME and "CHAR" are usually used within the database system.	Not recommended
JSON	JSON data does not support operators.	Not recommended
HLL	You are advised to use the HLL functions to reduce the impact on performance.	Not recommended
Currency	The MONEY type stores a currency amount with fixed fractional precision.	Not recommended
Geometric	POINT, LSEG, BOX, PATH, POLYGON, and CIRCLE	Not recommended
Network address	Stores IPv4 and MAC addresses.	Not recommended

- [Rule] Use the most specific numeric data types. If all of the following numeric types provide the required service precision, they are recommended in descending order of priority: integer, floating point, and NUMERIC.
- [Rule] Properly set the data type of a numeric column based on the value range, and use the NUMERIC or DECIMAL type as less as possible. NUMERIC and DECIMAL are equivalent. NUMERIC or DECIMAL data operations consume great CPU resources.

Table 4-8 Storage space and value range of numeric data types

Type	Storage Size (Unit: Byte)	Minimum Value	Maximum Value
TINYINT	1	0	255
SMALLINT	2	-32768	32767
INTEGER	4	-2,147,483,648	2,147,483,647
BIGINT	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
REAL/FLOAT4	4	6-bit decimal digits	

Type	Storage Size (Unit: Byte)	Minimum Value	Maximum Value
DOUBLE PRECISION/ FLOAT8	8	15-bit decimal digits	

- [Rule] Select a proper string type. If the value of a column must be a fixed-length character, use fixed-length character types or automatically add spaces. Otherwise, use the variable-length character type VARCHAR.

For a typical fixed-length column, for example, **gender**, you can enter only **f** or **m** that occupies a byte. You are advised to use the fixed-length data type (for example, CHAR(*n*)) for this type of columns.

If such requirement does not exist or longer characters may be required for future expansion, use variable-length character types (such as VARCHAR and TEXT) preferentially. You are advised not to specify the length of variable-length characters.

The reasons are as follows:

 - For fixed-length columns, the input data that is shorter than the fixed length will be padded with space characters and then be saved to the database. This wastes the storage space in the database.
 - For fixed-length character types, the entire table needs to be scanned and rewritten if the length needs to be extended later. This causes high performance overhead and affects online services.
 - For a variable-length column with a fixed length, the system checks whether the length exceeds the limit each time upon data insertion. This causes performance overhead.
- [Rule] Do not store data of the numeric type in columns of the character type.

If numeric calculation or comparison (for example, adding a filter condition) is performed on data stored in columns of the character type, unnecessary overhead will be caused due to data type conversion, and the column indexes may become invalid, affecting query performance.
- [Rule] Do not store data of the time or date type in columns of the character type.

If calculation or comparison (for example, adding a filter condition) with data of the time or date type is performed on data stored in columns of the character type, unnecessary overhead will be caused by data type conversion, and the column indexes may become invalid, affecting query performance.
- [Rule] Add NOT NULL constraints to columns that never have NULL values.

In certain scenarios, the optimizer may specially optimize **NOT NULL** columns to improve query performance.
- [Rule] Use the same data type for joined columns.

If the column types are inconsistent during a join operation, overhead will be caused by data type conversion.

- [Rule] The number of large fields (such as varchar (1000) and varchar (4000)) is not to exceed eight.
- [Recommendation] When defining a column, you are advised to create a comment for the column to facilitate subsequent maintenance.
For details about the description, value range, and usage of different types of fields, see [Data Type](#).
- [Recommendation] In tables that are logically related, columns having the same meaning should use the same data type.
- [Recommendation] For string data, you are advised to use variable-length strings and specify the maximum length. To avoid truncation, ensure that the specified maximum length is greater than the maximum number of characters to be stored. You are advised not to use CHAR(n), BPCHAR(n), NCHAR(n), or CHARACTER(n), unless you know that the string length is fixed.
- [Recommendation] Add NOT NULL constraints to columns that are used for WHERE filtering and join operations.
In certain scenarios, the optimizer may specially optimize NOT NULL columns to greatly improve query performance.
- [Recommendation] Do not reserve columns for a table. In most cases, you can quickly add or delete table columns, or change the default values of columns.
An added column must meet the following requirements. Otherwise, the entire table is updated, leading to additional overheads and affecting online services.
 - a. The data type is BOOLEAN, BYTEA, SMALLINT, BIGINT, SMALLINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, or INTERVAL.
 - b. The length of the default value cannot exceed 128 bytes.
 - c. The default value of the added column does not contain the volatile function.
 - d. The default value is required and cannot be **NULL**.

If you are not sure whether the third condition is met, contact GaussDB technical support for evaluation.

4.2.9 Index Design Specifications

- [Specification] Use the index types in the recommended database index practices.

 **NOTE**

Recommended types must be used for index design.

Table 4-9 Recommended database index practices

Index Type	Description	Recommended or Not
Primary key or unique index	Single-column or multi-column primary key or unique index.	Recommended

Index Type	Description	Recommended or Not
Global index	Index organization method.	In planning
Expression index	An index column is a function or scalar expression calculated from one or multiple columns in a table.	Restricted
B-tree index	Index building type.	Recommended for some data types

- [Rule] A composite index contains a maximum of 5 columns.
- [Rule] A composite index contains a maximum of 200 characters.
- [Rule] Index (including single-column index and composite index) columns should be **NOT NULL**.
- [Recommendation] Do not create multiple unique indexes for a single table. Maintaining multiple unique indexes at the same time generates more overheads than maintaining a multi-column unique index. If multiple unique indexes are equivalent to a multi-column unique index in service logic, a multi-column unique index shall be preferred.
- [Recommendation] The efficiency of maintaining indexes created for the same column is different. Columns of the number type are better than those of the character type and other data types. Therefore, it is recommended that columns such as IDs and time for creating indexes be stored as data of the number type.
- [Recommendation] Create indexes on joined columns.
HASH JOIN is supported, whereas **NESTLOOP JOIN** may be used for join operations if the rescan cost is low (for example, the inner table is small). If the **NESTLOOP JOIN** plan can be viewed by executing **EXPLAIN**, you can create indexes on joined columns to improve the efficiency of executing **NESTLOOP JOIN**.
- [Recommendation] When creating a composite index, you need to create it based on query conditions and the leftmost match principle of the composite index.

 NOTE

- Leftmost match principle of a composite index: If a query condition contains one or more columns of a composite index, the leftmost consecutive columns of the composite index must match the query condition.
- When the query is **WHERE a = ?, b = ?, c = ?, d = ?** or **WHERE a = ?, b = ?, c = ?, d = ?** or **WHERE b = ?, c = ?, d = ?** or **WHERE c = ?, d = ?**, the **idx_test_abcd** index might still be used after cost calculation. In such cases, the index scan will involve all pages of the index, resulting in unsatisfactory SQL performance. In similar cases, you are advised to create a composite index suitable for the query condition based on the leftmost match principle.

```
-- Create the test table.
gaussdb=# CREATE TABLE test(a int, b int, c int, d int, e int, f text);
Create a composite index.
gaussdb=# CREATE INDEX idx_test_abcd ON test(a,b,c,d);
```

4.2.10 Function/Stored Procedure Design Specifications

- [Rule] Do not use stored procedures or triggers to implement service logic. Instead, process the logic on the service server to prevent logical dependency on the database.
- [Rule] Do not use stored procedures to implement the upgrade logic in the database upgrade scripts of a service.
- [Rule] Create functions that have fixed return values for fixed input parameters and set the function type to **IMMUTABLE** or **SHIPPABLE**.

Currently, the database supports three types of functions: **IMMUTABLE**, **STABLE**, and **VOLATILE**. If the attribute of an **IMMUTABLE** function is set to **SHIPPABLE**, the function can be executed on DNs. In most scenarios, such functions perform efficiently. However, such functions require fixed return values for fixed input parameters to ensure that the functions are correctly executed on DNs. If the function execution result depends on the table scan result (for example, maximum value of a column in the table) or scan time (for example, current time), set the function type to **STABLE** or **VOLATILE**, and set the function attribute to **NOT SHIPPABLE** to ensure that the function is correctly executed. In this scenario, data on all DNs is sent to one CN for computing, resulting in low query efficiency.

4.2.11 Constraint Design

DEFAULT and NULL Constraints

- [Recommendation] If all the column values can be obtained from services, you are advised not to use the **DEFAULT** constraint. Otherwise, unexpected results will be generated during data loading.
- [Recommendation] Add **NOT NULL** constraints to columns that never have **NULL** values. The optimizer automatically optimizes the columns in certain scenarios.
- [Recommendation] Explicitly name all constraints excluding **NOT NULL** and **DEFAULT**.

Unique Constraints

- [Recommendation] The constraint name should indicate that it is a unique constraint, for example, **UNI***Included columns*.

Primary Key Constraints

- [Recommendation] The constraint name should indicate that it is a primary key constraint, for example, **PK***Column name*.

Check Constraints

- [Recommendation] The constraint name should indicate that it is a check constraint, for example, **CK***Column name*.

4.2.12 View and Joined Table Design

View Design

- [Recommendation] Do not nest views unless they have strong dependency on each other.
- [Recommendation] Try to avoid collation operations when defining a view.

Joined Table Design

- [Recommendation] Minimize joined columns across tables.
- [Recommendation] Use the same data type for joined columns.
- [Recommendation] The names of joined columns should indicate their relationship. For example, they can use the same name.

4.3 Database Programming Specifications

4.3.1 GUC Parameter Programming Specifications

[Rule] Clients (such as JDBC) should use default parameters for query. If session-level GUC parameters need to be modified, exercise caution when performing this operation.

When modifying GUC parameters through ODBC or JDBC, note that the GUC parameters take effect only in the current connection. Especially in the connection pool scenario, problems may occur and cause difficulty in locating problems.

If the GUC parameters must be set in a connection, you must use the following statements before releasing the connection to the connection pool:

```
SET SESSION AUTHORIZATION DEFAULT;  
RESET ALL;
```

Clear the connection status.

4.3.2 Object Access Programming Specifications

- [Rule] When operating an object, a user should have the operation permission on the object.

For details about the permission description, see [Users and Permissions](#) and comply with [Permission Design Specifications](#).

- [Rule] Use *schemaname.tablename* to access objects (such as tables and functions).

If the schema name prefix is not added, the system searches all tablespaces in the current **search_path** until a matched table is found, which causes unnecessary performance overhead.

4.3.3 WHERE

- [Rule] Do not compare table columns with the same WHERE condition.

For example, the following statement does not meet specifications:

```
SELECT * FROM t1 WHERE col1 = col1;
```

The following modification should be considered:

```
SELECT * FROM t1 WHERE col1 IS NOT NULL;
```

- [Rule] Do not include implicit data type conversion in WHERE conditions.

After implicit conversion is performed in the database, the created index may fail to be used.

You are advised to enable the GUC parameter **check_implicit_conversions** and disable **enable_fast_query_shipping** during development to check whether query statements contain implicit data types that may affect performance.

```
SET enable_fast_query_shipping = off;  
SET check_implicit_conversions = true;
```

Implicit data type conversion check requires extra overhead. After the query statement is developed, disable the **check_implicit_conversions** parameter and reset **enable_fast_query_shipping**.

Example:

- The following codes do not meet specifications:

```
-- The phonenummer column in the t_tablename table is of the VARCHAR type instead of the  
NUMERIC type.  
SELECT column1  
  INTO i_l_variable1  
  FROM t_tablename  
 WHERE phonenummer = 13512345678;
```

- The following modification should be considered:

```
-- The phonenummer column in the t_tablename table is of the VARCHAR type instead of the  
NUMERIC type.  
SELECT column1  
  INTO i_l_variable1  
  FROM t_tablename  
 WHERE phonenummer = '13512345678';
```

- [Rule] Do not use expressions or functions in WHERE condition columns.

When expressions or functions are used in condition columns, indexes become invalid, and each row of data is calculated, causing unnecessary performance consumption.

Example:

- The following codes do not meet specifications:

```
SELECT income FROM table WHERE abs(income) > ?;  
SELECT income FROM table WHERE income * 10 > ?;  
SELECT create_time  
  FROM table  
 WHERE date_format(create_time, '%Y%m%d %H:%i:%s') = '20090101 00:00:0';
```

- The following modification should be considered:

```
SELECT income FROM table WHERE income > ? OR income < (-1) * ?;
SELECT income FROM table WHERE income > ?/10;
SELECT create_time
FROM table
WHERE create_time = str_to_date('20090101 00:00:0', '%Y%m%d %H:%i:%s');
```

- [Rule] During table query, the WHERE condition must contain all partitioning keys. Otherwise, the query will be performed on multiple nodes, affecting the system concurrency and performance.
- [Rule] Do not use comparison operator (!=) when comparing with NULL in query conditions. Instead, use IS NULL or IS NOT NULL.
- [Rule] Do not use comparison operator (!=) in the index columns of the query condition to avoid invalid indexes.
- [Rule] Do not compare the index columns of the query condition with NULL (IS NULL and IS NOT NULL).
- [Rule] Do not use NOT in the index columns of the query condition.
- [Rule] Do not use NOT IN in the index columns of the query condition.
- [Rule] Do not place % at the beginning of the LIKE statement for fuzzy query. If % is placed at the beginning of the LIKE statement, the index cannot be used and the entire table will be scanned.
- [Recommendation] The number of IN clauses in the WHERE statement shall be less than or equal to 500.
 - During the query, the values of all the IN clauses will be compared to check whether they are equal, which increases the overhead.
 - If the included values are relatively fixed, consider creating a REPLICATION table, writing the clause data into the table, and then using INNER JOIN to implement the inclusion query.
- [Recommendation] If the IN clause in the WHERE condition is not a constant but a column in the table, you are advised to change it to a subquery.

In this case, it is actually an unequal JOIN, which is executed through the nestloop plan. If the table is too large, the execution efficiency is low. You are advised to change the query to a subquery of the JOIN type.

Example:

- The following code does not meet specifications:

```
SELECT col1, COALESCE(max(col2 - 1), 0)
FROM t1, t2
WHERE t1.col1 = ANY(VALUES(id1), (id2))
GROUP BY col1;
```

- The following modification should be considered:

```
SELECT col1, COALESCE(max(tmp), 0) FROM
(
(
SELECT col1, (col2-1) AS tmp
FROM t1, t2
WHERE t1.col1 = t2.id1 AND t1.col1 != t2.id2
) UNION ALL (
SELECT col1, (col2-1) AS tmp
FROM t1, t2
WHERE t1.col1 = t2.id2
)
) GROUP BY col1;
```

- [Recommendation] Use equal operators, instead of not equal operators.

If a not equal operator (IN, BETWEEN, <, <=, >, or >=) is used in the WHERE condition, no index can be used in the following conditions because two range conditions cannot be used at the same time.

4.3.4 SELECT

- [Rule] Do not use the wildcard character (*) in the SELECT statement.
If the table structure is changed due to service or database upgrade when a wildcard column is used to query a table, the table structure may be incompatible with service statements. Therefore, the service must specify the name of the table column to be queried and avoid using the wildcard character.
- [Rule] SELECT LIMIT statements must contain ORDER BY for query.
As a distributed database, distributes table data across multiple DNs.
If the SQL statement contains only LIMIT but not ORDER BY, the database sends the result (meeting the query requirements) sent by the DN with fast network transmission to the client as the final result. The network transmission efficiency may change at different time points. As a result, the returned results are inconsistent when the SQL statement is executed for multiple times.
- [Rule] Do not perform operations that may cause sorting, such as ORDER BY, DISTINCT, GROUP BY, and UNION, on large columns (such as VARCHAR(2000)).
These operations consume a large number of CPU and memory resources, resulting in low execution efficiency.
- [Rule] Do not use the LOCK TABLE statement to lock a table. Instead, use the SELECT .. FOR UPDATE statement.
LOCK TABLE provides multiple levels of locks. However, if you do not fully understand the database principles and services, misuse of table locks may trigger deadlocks, causing the cluster to be unavailable.
- [Recommendation] Do not use subqueries in the SELECT target columns.
Otherwise, the plan may fail to be pushed down to DNs for execution, affecting the execution performance.
- [Recommendation] Use UNION ALL instead of UNION. Consider data deduplication if necessary.
UNION ALL does not deduplicate data and does not require sorting operations. Therefore, UNION ALL is faster than UNION. If deduplication is not required, UNION ALL is preferred.
- [Recommendation] Do not frequently use count() to obtain the number of rows in a large table. This operation consumes a large number of resources and affects the execution efficiency of parallel jobs.
If you do not need the real-time row statistics, run the following statement to obtain the number of rows in the table:

```
SELECT reltules FROM pg_class WHERE relname = 'tablename';
```


The number of rows recorded in pg_class is updated only after ANALYZE is executed on the table.
Currently, ANALYZE is triggered in either of the following conditions:
 - The service sends an ANALYZE statement. For example:

```
-- Analyze all tables in the connection library.  
ANALYZE;  
-- Analyze the specified table.  
ANALYZE tablename;
```

- This event is triggered when the number of rows added or deleted at a specified interval or in a table reaches a specified value by using the AUTO VACUUM mechanism. The interval and addition/deletion ratio can be set through the GUC parameters.

4.3.5 INSERT

- [Specification] INSERT ON DUPLICATE KEY UPDATE does not support UPDATE on columns with PRIMARY KEY or UNIQUE constraints.

The semantics of INSERT ON DUPLICATE KEY UPDATE is to update the rows that have UNIQUE constraint conflicts. In this process, the value of the constraint should not be updated.

- [Rule] Do not run INSERT ON DUPLICATE KEY UPDATE on a table that has multiple UNIQUE constraints.

A table may have multiple unique indexes, or both primary keys and unique indexes. When multiple UNIQUE constraints exist, the system checks all the UNIQUE constraints by default. If any constraint conflicts, the system updates the conflicting rows. That is, multiple records may be updated, which does not meet the service expectation. More specific conditions for inserting and updating data shall be added.

- [Recommendation] In the case of batch insertion, you are advised to use executeBatch to execute **INSERT INTO VALUES (?)**. The execution efficiency is higher than that of executing multiple **INSERT INTO VALUES()** or **INSERT INTO VALUES(?), ..., (?)** statements.

4.3.6 UPDATE

- [Specification] In GTM-free mode, cross-node transactions are not allowed. Therefore, when updating a data table distributed by hash, you must specify the equal condition in the WHERE condition for the distribution keys.
- [Rule] The UPDATE statement must contain the WHERE clause to avoid full table scan.
- [Rule] Do not use the updated columns as the update source when the UPDATE clause updates multiple columns simultaneously.

Even if multiple columns are updated simultaneously from the same source, the behavior varies depending on the database. To avoid compatibility issues, avoid the preceding operations at the service layer. Example:

```
UPDATE table SET col1 = col2, col3 = col1 WHERE col1 = 1;
```

In , the value of **col3** is the original value of **col1**. In MySQL, the value of **col3** is the value of **col2** (because the value of **col2** is assigned to **col1**).

- [Rule] Do not use ORDER BY or GROUP BY in the UPDATE statement to avoid unnecessary sorting.
- [Recommendation] If a table has a primary key or index, the WHERE condition must be used together with the primary key or index during update.

4.3.7 DELETE

- [Specification] In GTM-free mode, cross-node transactions are not allowed. Therefore, when deleting data of a HASH-distributed table, you must specify the equal condition in the WHERE condition for the distribution keys.
- [Rule] The DELETE statement must contain the WHERE clause to avoid full table scan.
- [Rule] Do not use ORDER BY or GROUP BY in the DELETE statement to avoid unnecessary sorting.
- [Rule] Use TRUNCATE instead of DELETE to clear a table.
TRUNCATE creates a new physical file and physically deletes the original file when the transaction ends to clear the disk space. However, the DELETE statement marks data in the table and does not clear the disk space until the VACUUM FULL phase.
- [Recommendation] If **DELETE** is executed on a table that has a primary key or index, the WHERE condition must be used together with the primary key or index to improve execution efficiency.

4.3.8 Join Query

- [Rule] The nesting depth of multi-table join must be less than 8.
If the join nesting is too deep, slow SQL statements may be generated. You need to optimize the join nesting at the service layer.
- [Rule] Specify the join condition (ON) of each table to avoid Cartesian product.
For example, in MySQL, JOIN is equivalent to CROSS JOIN and INNER JOIN. However, in the SQL standards, JOIN is equivalent to INNER JOIN only. Therefore, the ON condition must be used together.
- [Rule] Specify the JOIN mode based on the SQL standards during join. Do not use the JOIN keyword directly. Instead, use CROSS JOIN, INNER JOIN, LEFT JOIN, or RIGHT JOIN.
- [Rule] When multiple tables are joined for query, aliases must be added to the tables to ensure that the statement logic is clear and easy to maintain.
- [Recommendation] Different columns have different comparison overheads. Use a column type with high efficiency for joined columns.
 - The comparison efficiency of the numeric type is much higher than that of the string type.
 - The comparison efficiency of integers is much higher than that of numeric and floating-point types.
- [Recommendation] The joined columns must be of the same data type to avoid the impact of implicit type conversion on the execution efficiency.
- [Recommendation] Do not use nested subqueries. Use table join if possible because subqueries will generate temporary tables, which greatly affects SQL performance.
- [Recommendation] If a large number of NULL values exist in the joined columns, you are advised to add the IS NOT NULL condition to the WHERE condition to improve the execution efficiency.

4.3.9 Subquery

- [Rule] Do not use repeated subquery statements in an SQL statement.
- [Recommendation] Do not use scalar subqueries.

A scalar subquery is a subquery whose result is one value and whose condition expression uses an equal operator. Example:

The following statement does not meet specifications:

```
SELECT * FROM t1 WHERE id = (SELECT id FROM t2 LIMIT 1);
```

You are advised to split the preceding statement into two SQL statements and execute the subquery first.

- [Recommendation] Do not use subqueries in the SELECT target columns. Otherwise, the plan may fail to be pushed down, affecting the execution performance.
- [Recommendation] The nesting depth of a subquery cannot exceed 2. Subqueries cause temporary table overhead. Therefore, complex queries must be optimized based on service logic.

4.3.10 Transaction

- [Specification] In GTM-free mode, cross-node transactions are not allowed.

In GTM-free mode, if an executed SQL statement contains a cross-node transaction, the error message "Unsupport DML two phase commit under gtm free mode." is displayed.

In this case, if a statement must temporarily start a cross-node transaction, run the following command in the transaction to start the cross-node transaction:

```
SET enable_twophase_commit = on;
```

Before the transaction ends, run the following command:

```
RESET enable_twophase_commit;
```

Or:

```
SET enable_twophase_commit = off;
```

- [Specification] Large object operations do not support transactions. Large object operations include CREATE/DELETE DATABASE, ANALYZE, and VACUUM jobs.
- [Rule] When accessing the database through the JDBC client, disable the **autocommit** parameter and explicitly execute the transaction COMMIT.
 - On the one hand, enabling the **autocommit** parameter will cause some parameters (such as **fetchsize**) to become invalid.
 - On the other hand, the service should clarify the service logic and reduce the dependence on the database.
- [Rule] Do not combine multiple SQL statements into one statement when accessing the database through JDBC.

If multiple statements are combined into one, as long as one of them fails to be executed, the whole statement fails and a failure message is returned. This is inconvenient for fault locating. You are advised to split the statement.

Example:

- The following statement does not meet specifications:

```
Connection conn = ....
try {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();
        stmt.executeUpdate("CREATE TABLE t1 (a int); DROP TABLE t1");
    } finally {
        stmt.close();
    }
    conn.commit();
} catch(Exception e) {
    conn.rollback();
} finally {
    conn.close();
}
```

- You are advised to split the statement into two statements and send them separately:

```
Connection conn = ....
try {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();
        stmt.executeUpdate("CREATE TABLE t1 (a int)");
        stmt.executeUpdate("DROP TABLE t1");
    } finally {
        stmt.close();
    }
    conn.commit();
} catch(Exception e) {
    conn.rollback();
} finally {
    conn.close();
}
```

4.3.11 SQL Compilation

DDL

- In GaussDB, you are advised to perform DDL operations (such as table creation and COMMENT) in a unified manner. Prevent DDL operations in batch processing jobs so that performance is not affected.
- Perform the TRUNCATE operation immediately after the unlogged table is used. GaussDB cannot ensure the security of unlogged tables in abnormal scenarios.
- Suggestions on the storage mode of temporary and unlogged tables are the same as those on base tables.
- The total length of an index column cannot exceed 50 bytes. Otherwise, the index size will increase greatly, resulting in large storage cost and low index performance.
- Do not delete objects using DROP...CASCADE, unless the dependency between objects is specified. Otherwise, the objects may be deleted by mistake.

Data Loading and Unloading

- Explicitly set the inserted column list in the INSERT statement. Example:
INSERT INTO task(name,id,comment) VALUES ('task1','100','100th task');
- After data is imported to the database in batches or the data increment reaches the threshold, you are advised to analyze tables to prevent the execution plan from being degraded due to inaccurate statistics.

- To clear all data in a table, you are advised to use TRUNCATE TABLE instead of DELETE TABLE. DELETE TABLE is not efficient and cannot release disk space occupied by the deleted data.

Type Conversion

- Convert data types explicitly. If you perform implicit conversion, the result may differ from expected.
- During data query, explicitly specify the data type for constants, and do not attempt to perform any implicit data type conversion.
- If **sql_compatibility** is set to **ORA**, null strings will be automatically converted to NULL during data import. If null strings need to be retained, set **sql_compatibility** to **TD**.

Query Operation

- Do not return a large number of result sets to a client except the ETL program. If a large result set is returned, consider modifying your service design.
- Perform DDL and DML operations encapsulated in transactions. For example, operations such as TRUNCATE TABLE, UPDATE TABLE, DELETE TABLE, and DROP TABLE cannot be restored once they are committed. You are advised to encapsulate such operations in transactions so that you can roll back the operations if necessary.
- During query compilation, you are advised to list all columns to be queried and avoid using SELECT *. Doing so reduces output columns, improves query performance, and avoids the impact of adding or deleting columns on front-end service compatibility.
- During table object access, add the schema prefix to the table object to avoid accessing an unexpected table due to schema switchover.
- The cost of joining more than three tables or views, especially full joins, is difficult to be estimated. You are advised to use the WITH TABLE AS statement to create interim tables to improve the readability of SQL statements.
- Avoid using Cartesian products or full joins. Cartesian products and full joins will result in a sharp expansion of result sets and poor performance.
- Only IS NULL and IS NOT NULL can be used to determine NULL value comparison results. If any other method is used, NULL is returned. For example, NULL instead of expected Boolean values is returned for NULL<>NULL, NULL=NULL, and NULL<>1.
- Do not use count(col) instead of count(*) to count the total number of records in a table. count(*) counts the NULL value (actual rows) while count(col) does not.
- While executing count(col), the number of NULL record rows is counted as 0. While executing sum(col), NULL is returned if all records are NULL. If not all the records are NULL, the number of NULL record rows is counted as 0.
- To count multiple columns using count(), column names must be enclosed in parentheses. For example, count ((col1, col2, col3)). Note: When multiple columns are used to count the number of NULL record rows, a row is counted even if all the selected columns are NULL. The result is the same as that when count(*) is executed.

- NULL records are not counted when count(distinct col) is used to calculate the number of non-NULL columns that are not repeated.
- If all statistical columns are NULL when count(distinct (col1,col2,...)) is used to count the number of unique values in multiple columns, Null records are also counted, and the records are considered the same.
- Use the connection operator || to replace the concat function for string connection, because the output of the concat function depends on the data type of the strings to be connected. When the execution plan is generated, the value cannot be calculated in advance. As a result, the query performance deteriorates severely.
- Use the time-related macros listed in [Table 1](#) to replace the now function and obtain the current time, because the execution plan generated by the now function cannot be pushed down to disks. As a result, the query performance severely deteriorates.

Table 4-10 Time-related macros

Macro Name	Description	Example
CURRENT_DATE	Obtains the current date, excluding the hour, minute, and second details.	gaussdb=# SELECT CURRENT_DATE; date ----- 2018-02-02 (1 row)
CURRENT_TIME	Obtains the current time, excluding the year, month, and day.	gaussdb=# SELECT CURRENT_TIME; timetz ----- 00:39:34.633938+08 (1 row)
CURRENT_TIMESTAMP(n)	Obtains the current date and time, including year, month, day, hour, minute, and second. NOTE <i>n</i> indicates the number of digits after the decimal point in the time string.	gaussdb=# SELECT CURRENT_TIMESTAMP(6); timestampz ----- 2018-02-02 00:39:55.231689+08 (1 row)

- Do not use scalar subquery statements. A scalar subquery is a subquery in the output list of the SELECT statement. In the following example, "SELECT COUNT(*) FROM films f WHERE f.did = s.id" is a scalar subquery statement:
SELECT id, (SELECT COUNT(*) FROM films f WHERE f.did = s.id) FROM staffs_p1 s;
Scalar subqueries often result in query performance deterioration. During application development, scalar subqueries need to be converted into equivalent table associations based on the service logic.
- In WHERE clauses, the filter conditions should be collated. The condition that few records are selected for reading (the number of filtered records is small) is listed at the beginning.

- Filter conditions in WHERE clauses should comply with unilateral rules, that is, to place the column name on one side of a comparison operator. In this way, the optimizer automatically performs pruning optimization in some scenarios. The format is *col op expression*, where *col* indicates a table column, *op* indicates a comparison operator, such as = and >, and *expression* indicates an expression that does not contain a column name. Example:

```
SELECT id, from_image_id, from_person_id, from_video_id FROM face_data WHERE
current_timestamp(6) - time < '1 days'::interval;
```

The modification is as follows:

```
SELECT id, from_image_id, from_person_id, from_video_id FROM face_data where time >
current_timestamp(6) - '1 days'::interval;
```

- Do not perform unnecessary collation operations. Collation requires a large amount of memory and CPU. If service logic permits, ORDER BY and LIMIT can be combined to reduce resource overhead. By default, GaussDB performs collation by ASC & NULL LAST.
- When the ORDER BY clause is used for collation, specify collation modes (**ASC** or **DESC**), and use NULL FIRST or NULL LAST for NULL record sorting.
- Do not rely on only the LIMIT clause to return the result set displayed in a specific sequence. Combine ORDER BY and LIMIT clauses if some specific result sets are returned, and use OFFSET to skip specific results if necessary.
- If the service logic is accurate, you are advised to use UNION ALL instead of UNION.
- If a filter condition contains only an OR expression, convert the OR expression to UNION ALL to improve performance. SQL statements that use OR expressions cannot be optimized, resulting in slow execution. For example, the conversion of the following statements:

```
SELECT * FROM scdc.pub_menu
WHERE (cdp= 300 AND inline=301) OR (cdp= 301 AND inline=302) OR (cdp= 302 AND inline=301);
```

Convert the statement to the following:

```
SELECT * FROM scdc.pub_menu
WHERE (cdp= 300 AND inline=301)
union all
SELECT * FROM scdc.pub_menu
WHERE (cdp= 301 AND inline=302)
union all
SELECT * FROM tablename
WHERE (cdp= 302 AND inline=301)
```

- If an IN(val1, val2, val3...) expression contains a large number of columns, you are advised to replace it with the IN (VALUES (val1), (val2),(val3)...) statement. The optimizer will automatically convert the IN constraint into a non-correlated subquery to improve the query performance.
- Use (NOT) EXIST instead of (NOT) IN when associated columns do not contain null values. For example, in a query statement, if the **T1.C1** column does not contain any **NULL** value, add the **NOT NULL** constraint to the **T1.C1** column, and then rewrite the statements.

```
SELECT * FROM T1 WHERE T1.C1 NOT IN (SELECT T2.C2 FROM T2);
```

Rewrite the statement as follows:

```
SELECT * FROM T1 WHERE NOT EXISTS (SELECT * FROM T2 WHERE T1.C1=T2.C2);
```

 **NOTE**

- If the value of the **T1.C1** column is not **NOT NULL**, the preceding rewriting cannot be performed.
- If the **T1.C1** column is the output of a subquery, check whether the output is **NOT NULL** based on the service logic.
- Use cursors instead of the **LIMIT OFFSET** syntax to perform pagination queries to avoid resource overheads caused by multiple executions. A cursor must be used in a transaction, and you must disable the cursor and commit the transaction once the query is finished.

4.3.12 DDL

- [Rule] Do not perform DDL operations during peak hours. If DDL operations must be performed, control the DDL operation execution frequency (no more than one DDL operation per second).
 - a. DDL locks are managed using a global regular lock table. DDL operations compete for access to the regular lock table, resulting in a large number of threads being blocked in the LockMgrLock wait event and a risk of thread pool resource exhaustion. This may trigger a switchover or the thread pool's anti-overload mechanism, impacting the service success rate.
 - b. DDL operations require the system catalog cache for invalid objects, enabling other concurrent threads to detect changes in DDL objects. Moreover, other threads need to handle invalid messages, which increases thread processing pressure and poses a risk of increased CPU usage.
 - c. The replay speed of DDL logs is significantly slower than that of DML logs. Frequent DDL operations may increase latency for both primary and standby nodes, triggering flow control. Therefore, it is recommended to control the DDL operation frequency during peak hours to avoid triggering flow control.
- [Rule] DDL and other service operations cannot be performed in the same transaction; DDL operations must be executed in an independent transaction.

4.4 Client Programming Specifications

4.4.1 JDBC

- When a third-party tool connects to GaussDB through JDBC, JDBC sends a connection request to GaussDB. By default, the following configuration parameters are added. For details, see the implementation of the `ConnectionFactoryImpl` class in the JDBC code.

```
params = {  
    { "user", user },  
    { "database", database },  
    { "client_encoding", "UTF8" },  
    { "DateStyle", "ISO" },  
    { "extra_float_digits", "3" },  
    { "TimeZone", createPostgresTimeZone() },  
};
```

These parameters may cause the JDBC and gsql clients to display inconsistent data, for example, date data display mode, floating point precision representation, and timezone.

If the result is not as expected, you are advised to explicitly set these parameters in the Java connection setting.

When the database is connected through JDBC, **extra_float_digits** is set to **3**. When the database is connected using gsql, **extra_float_digits** is set to **0**. As a result, the precision of the same data displayed in JDBC clients may be different from that displayed in gsql clients.

- [Specification] A database must be specified for each JDBC instance. Once the instance is created, it cannot switch to another database.
- [Specification] The length of a single SQL statement cannot exceed 2 GB. Considering the communications cost, it is recommended that the length of a single SQL statement be less than or equal to 5 KB.
- [Specification] Currently, only the DEFAULT value of the columns in CREATE/ALTER TABLE can be set through parameters. Other DDL statements do not support parameter settings using the Prepare Execute method.
- [Rule] Each PreparedStatement of JDBC can contain a maximum of 32767 parameters.
- [Specification] The connection parameter **fetchsize** must be used when **autocommit** is disabled. Otherwise, the **fetchsize** configuration is invalid.
- [Rule] Use the default GUC parameters and avoid sending SET requests through JDBC to modify GUC parameters.

For details, see [GUC Parameter Programming Specifications](#).

- [Rule] You must use the Prepare Execute method to execute query statements to improve execution efficiency.
- [Rule] Execute SQL statements one by one in the same transaction. Do not combine multiple SQL statements and send them as one statement.

For details, see [\[Rule\] Do not combine multiple SQL statements into one statement when accessing the database through JDBC](#).

- [Rule] The time zone of the host where the JDBC client is located, the time zone of the host where the database cluster is located, and the time zone during cluster configuration must be the same.
- [Rule] If a temporary table is created in a connection, delete the temporary table before releasing the connection to the connection pool to avoid service errors.
- [Recommendation] In precision-sensitive scenarios, the numeric type is recommended.
- [Recommendation] When connecting to the database through JDBC, ensure that the following three time zones are the same:
 - Time zone of the host where the JDBC client is located
 - Time zone of the host where the GaussDB cluster is located
 - Time zone during GaussDB cluster configuration

NOTE

For details about how to set the time zone, contact the administrator.

- [Recommendation] Enable **autocommit** in the code for connecting to GaussDB by the JDBC. If **autocommit** needs to be disabled to improve performance or for other purposes, applications need to ensure that transactions are committed. For example, explicitly commit transactions after specifying service SQL statements. Particularly, ensure that all transactions are committed before the client exits.
- [Recommendation] You are advised to use connection pools to limit the number of connections from applications. You are advised not to connect to a database each time an SQL statement is executed.
- [Recommendation] After an application completes its jobs, disconnect it from GaussDB to release occupied resources. You are advised to set the session timeout interval in the jobs.
- [Recommendation] Reset the session environment before releasing connections to the JDBC connection tool. Otherwise, historical session information may cause object conflicts.
 - If GUC parameters are set in the connection, run **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to clear the connection status before you return the connection to the connection pool.
 - If a temporary table is used, delete the temporary table before you return the connection to the connection pool.
- [Recommendation] In the scenario where the ETL tool is not used and real-time data import is required, it is recommended that you use the CopyManager API driven by the GaussDB JDBC to import data in batches during application development.
- [Recommendation] Set **prepareThreshold** to a proper value. If the query statement is fixed, set it to **1**.
- [Recommendation] You are advised to set the connection parameter **autoBalance** to **true** to enable the CN load balancing function and set multiple CN connection addresses (separated by commas).
 - Once **autobalance** is enabled, the JDBC driver attempts to allocate JDBC connections to different CNs.
 - The purpose of setting multiple CN connection addresses is to prevent the JDBC driver from failing to obtain the CN list of the cluster for the first time due to a CN fault.
 - Once the CN list is obtained successfully for the first time, the system does not depend on the CN list specified in the connection parameters. Instead, the system obtains the latest CN list by connecting to a valid CN in the cluster CN list obtained in real time at a specified interval.
- [Recommendation] You are advised to set **batchMode** to **on** to use the batch connection mode to improve the execution performance.
- [Recommendation] Set a proper JDBC connection timeout interval based on the upper-layer request timeout of the service to prevent the job from always occupying database resources.

Timeout parameters include **loginTimeout**, **connectTimeout**, and **socketTimeout**.

 - **loginTimeout**: integer type. This parameter specifies the waiting time for establishing the database connection, in seconds. The default value is **0**, indicating that the timeout mechanism is disabled.

- **connectTimeout**: integer type. This parameter specifies the timeout interval for connecting to a server. If the time taken to connect to a server exceeds the value specified, the connection is interrupted. The unit of the timeout interval is second. The default value **0** indicates that the timeout mechanism is disabled.
- **socketTimeout**: integer type. This parameter indicates the timeout interval for a socket read operation. If the time taken to read data from a server exceeds the value specified, the connection is closed. The unit of the timeout interval is second. The default value **0** indicates that the timeout mechanism is disabled.
- **cancelSignalTimeout**: integer type. A cancel message may cause a block. This attribute controls "connect timeout" and "socket timeout" in a cancel command. The default value is **10**.
- **tcpKeepAlive**: Boolean type. This parameter is used to enable or disable TCP keepalive detection. The default value is **false**.

4.4.2 Connection Control Specifications

- [Rule] Concurrency control is required to avoid exceeding the maximum number of CN connections multiplied by the number of CNs, or the maximum number of connections on a single DN, thus preventing over-limit connection errors on CNs or DNs. If frequent over-limit connection errors occur (more than 10 errors per second), it may lead to service performance deterioration, significantly increased thread pool and CPU usage, potential zombie detection timeout, abnormal cluster status, and even a switchover.
- [Rule] It is recommended to maintain a static connection pool or regulate the frequency of establishing connections to prevent frequent creation of new ones. If the connection frequency of a single CN or DN exceeds 10 times per second, it may lead to a sharp increase in thread pool and CPU usage, thereby significantly impacting service performance.

5 Application Development Guide

5.1 GaussDB Application Development Guide

In GaussDB, you can connect to and perform operations on a database through APIs such as JDBC, ODBC, libpq, Psycopg, ECPG, and Go.

JDBC

Java Database Connectivity (JDBC) is a Java API for running SQL statements. It provides unified access APIs for different relational databases. It allows Java applications to use SQL statements to perform database operations, such as querying, inserting, updating, and deleting data.

Some key features and usage of JDBC are as follows:

1. Database connection management: JDBC allows applications to establish connections to databases and manages the lifecycle of these connections.
2. SQL execution: JDBC can be used to query, update, and delete SQL statements. This is achieved by constructing SQL statements in Java code and sending them to the database.
3. Transaction management: JDBC APIs can be used to start, commit, or roll back transactions to ensure the consistency and integrity of database operations.
4. Exception handling: JDBC defines a group of exception classes to handle various exceptions that may occur during database operations. Developers can use the try-catch block to capture and handle these exceptions.
5. Batch processing: JDBC provides the batch processing function, which allows multiple SQL statements to be executed concurrently, improving the efficiency of database operations.
6. Metadata access: JDBC can be used to obtain the metadata information of the database, such as the table structure, column name, and data type. Developers then can dynamically construct SQL query statements or perform operations based on the database structure.

In conclusion, JDBC provides a flexible and powerful bridge that enables Java applications to interact with various databases and to implement data persistence

and management. GaussDB supports JDBC 4.2 and requires JDK 1.8 for code compiling. It does not support JDBC-ODBC bridging.

For details about JDBC-based development, see [Development Based on JDBC](#).

ODBC

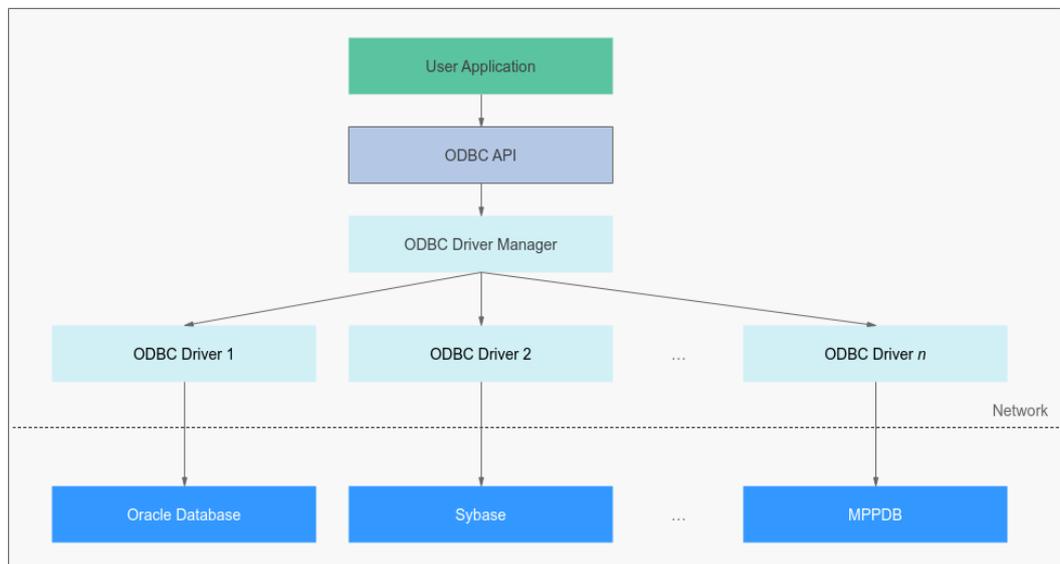
Open Database Connectivity (ODBC) is a Microsoft API for accessing databases in C/C++ based on the X/OPEN CLI. It provides a unified method for applications to access various database management systems (DBMSs) without considering specific database types or OS platforms. ODBC allows applications to use SQL to query, insert, update, and delete data in a database. Applications interact with the database through the APIs provided by ODBC, which enhances their portability, scalability, and maintainability.

The ODBC architecture consists of three main components: application, ODBC driver manager, and ODBC driver. Applications use ODBC APIs to communicate with the ODBC driver manager which loads and manages ODBC drivers. The ODBC drivers are responsible for communicating with a specific database, executing SQL queries, and returning results. For details about the ODBC system architecture, see [Figure 5-1](#).

In conclusion, ODBC provides a flexible and cross-platform method that allows users to easily connect applications to various databases without worrying about the details of a specific database system.

For details about ODBC-based development, see [Development Based on ODBC](#).

Figure 5-1 ODBC system architecture



GaussDB supports ODBC 3.5 in the following environments.

Table 5-1 OSs supported by ODBC

OS	Platform
EulerOS 2.5	x86_64
EulerOS 2.9	Arm64
EulerOS 2.10	x86_64
EulerOS 2.10	Arm64
Windows 7	x86_32
Windows 7	x86_64
Windows Server 2008	x86_32
Windows Server 2008	x86_64
Kylin V10	x86_64
Kylin V10	Arm64
UnionTech V20	x86_64
UnionTech V20	Arm64

The ODBC Driver Manager running on Unix or Linux can be unixODBC or iODBC. unixODBC-2.3.7 is used as the component for connecting to the database.

Windows has a native ODBC Driver Manager. You can locate **Data Sources (ODBC)** by choosing **Control Panel > Administrative Tools**.

NOTE

The current database ODBC driver is based on an open-source version and may be incompatible with Huawei-developed data types such as tinyint, smalldatetime, and nvarchar2.

ODBC limitations:

- ODBC does not support user-defined types and does not support user-defined parameters in stored procedures.
- When the **proc_outparam_override** parameter is enabled for the database, ODBC cannot properly call the stored procedure that contains the **out** parameter.
- ODBC does not support read on the standby node in a cluster for DR.

libpq

libpq is a C application programming API to GaussDB. libpq contains a set of library functions that allow client programs to send query requests to GaussDB servers and obtain query results. It is also the underlying engine of other GaussDB APIs, such as ODBC.

For details about libpq-based development, see [Development Based on libpq](#).

Psycopg

Psycopg is a Python API used to execute SQL statements and provides a unified access API for GaussDB. Applications can perform data operations based on psycopg. Psycopg2 is the encapsulation of libpq and is implemented using the C language, which is efficient and secure. It provides cursors on both client and server, asynchronous communication and notification, and the COPY TO and COPY FROM functions. It supports multiple types of Python out-of-the-box and adapts to GaussDB data types. Through the flexible object adaptation system, you can extend and customize the adaptation. Psycopg is compatible with Unicode.

GaussDB supports the psycopg2 feature and allows psycopg2 to be connected in SSL mode.

For details about Psycopg-based development, see [Psycopg-based Development](#).

Table 5-2 Platforms supported by psycopg

OS	Platform	Python Version
EulerOS 2.5	<ul style="list-style-type: none">• Arm64• x86_64	3.8.5
EulerOS 2.9	<ul style="list-style-type: none">• Arm64• x86_64	3.7.4
EulerOS 2.10, Kylin V10, and UnionTech20	<ul style="list-style-type: none">• Arm64• x86_64	3.7.9
EulerOS 2.11 and SUSE 12.5	<ul style="list-style-type: none">• Arm64• x86_64	3.9.11

NOTICE

During psycopg2 compilation, OpenSSL of GaussDB is linked. OpenSSL of GaussDB may be incompatible with OpenSSL of the OS. If incompatibility occurs, for example, "version 'OPENSSL_1_1_1f' not found" is displayed, use the environment variable `LD_LIBRARY_PATH` to isolate the OpenSSL provided by the OS and the OpenSSL on which GaussDB depends.

For example, when the application software `client.py` that calls psycopg2 is executed, the environment variable is explicitly assigned to the application software.

```
export LD_LIBRARY_PATH=/path/to/gaussdb/libs:$LD_LIBRARY_PATH python client.py
```

In the preceding command, `/path/to/psycopg2/lib` indicates the directory where the OpenSSL library on which GaussDB depends is located. Change it as required.

ecpg

Embedded SQL C Preprocessor for GaussDB Kernel (ECPG) is an embedded SQL preprocessor for C programs. An embedded SQL program consists of code written

in an ordinary programming language, in this case C, mixed with SQL commands in specially marked sections. To build the program, the source code (*.pgc) is first passed through the embedded SQL preprocessor, which converts it to an ordinary C program (*.c), and afterwards it can be processed by a C compiler. The converted ECPG program calls functions in the libpq library through the embedded SQL library (ecpglib), and communicates with a GaussDB Kernel server using the normal frontend-backend protocol.

Programs written for the embedded SQL API are normal C programs with special code inserted to perform database-related actions. This special code always has the form:

```
EXEC SQL ...;
```

These statements syntactically take the place of a C statement. Depending on the particular statement, they can appear at the global level or within a function. Embedded SQL statements follow the case-sensitivity rules of normal SQL code, and allow nested C code-style comments (part of the SQL standard). However, the C part of the program follows the standards of the C program and does not support nested comments.

For details about ECPG-based development, see [ECPG-based Development](#).

Go

Go is a language driver used to connect to and operate GaussDB. It provides some APIs for connecting to and operating GaussDB. You can use these APIs to perform operations such as query, insert, update, and delete.

Some functions and usage of a Go driver are as follows:

1. Database connection: A Go driver can connect to a database through db.Open. The connection string can be in URL or DSN format.
2. SQL execution and query: A Go driver provides APIs such as db.Exec and db.Query for SQL execution and query.
3. Transaction management: A Go driver provides the Tx API for transaction management. It implements methods such as starting, committing, and rolling back transactions to ensure the consistency and atomicity of database operations.
4. Metadata access: A Go driver provides the ColumnType API to query column attribute information in the database, including whether the column can be empty, database type name, length, and number of decimal places.
5. Reusability: A Go driver provides the Prepare method for preparing SQL statements. The SQL statements can be executed for multiple times. You only need to change the input parameters to improve database reusability.

For details about Go-based development, see [Development Based on the Go Driver](#).

5.2 Development Specifications

If the connection pool mechanism is used during application development, comply with the following specifications:

- If GUC parameters are set in the connection, run **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to clear the connection status before you return the connection to the connection pool.
- If a temporary table is used, delete the temporary table before you return the connection to the connection pool.

If you do not do so, the connection state in the connection pool will remain, which affects subsequent operations using the connection pool.

Table 5-3 describes the compatibility of application development drivers.

Table 5-3 Compatibility description

Driver	Compatibility Description
JDBC and Go	The new drivers are forward compatible with the database. To use the new features added to the driver and database, you must upgrade the database.
ODBC, libpq, Psycopg, and ecpg	The driver version must match the database version.

NOTICE

- Setting **behavior_compat_options** to '**proc_outparam_override**' is applicable only in A-compatible mode.
- In principle, you need to set the compatibility parameter after the database creation, instead of switching the parameters when using the database.
- To use the features in the following scenario, you need to upgrade the JDBC driver to 503.1 or later, enable the **s2** compatibility parameter, and set the validity check of sessiontimezone:

If the driver is used in a multi-thread environment:

The JDBC driver is non-thread-safe and does not guarantee that the connection methods are synchronized. The caller synchronizes the calls to the driver.

5.3 Obtaining the Driver Package

Obtaining the Driver Package

Download the required packages listed in **Table 5-4**.

Table 5-4 Driver package download list

Version	Download Address
V2.0-3.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

1. Upload the software package and verification package to the same directory on a Linux VM.
2. Run the following command to verify the integrity of the software package:

```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

If **OK** is displayed in the command output, the verification is successful.

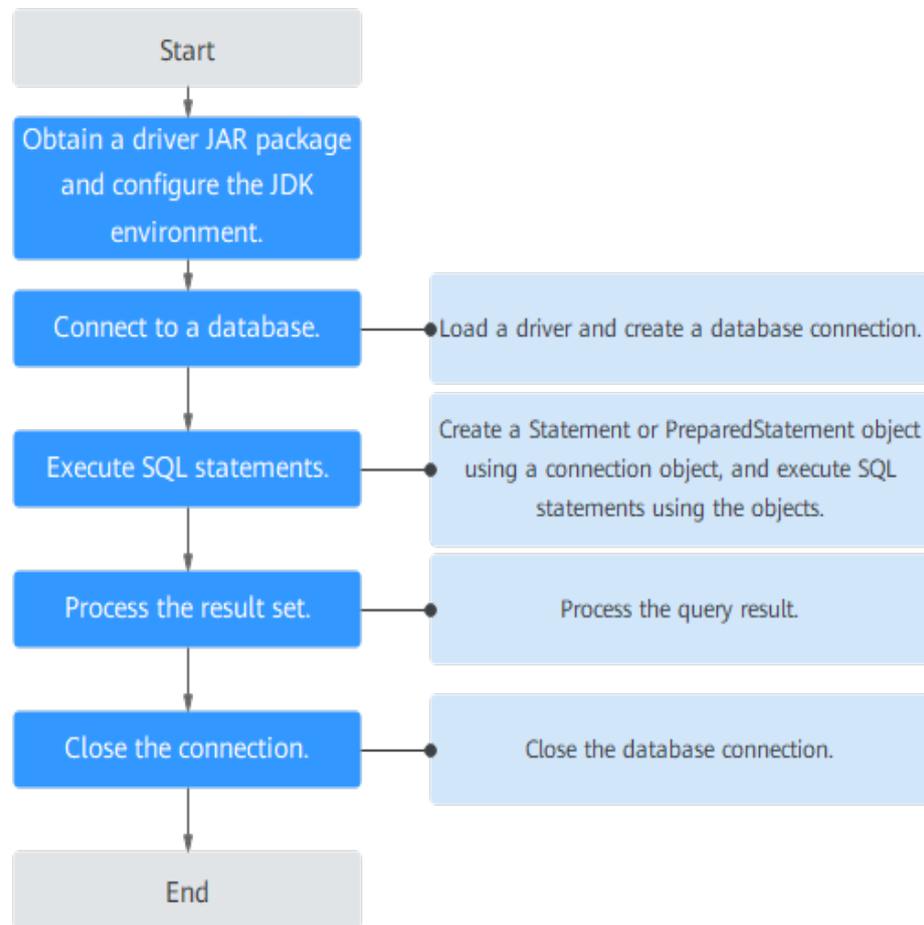
```
GaussDB_driver.zip: OK
```

5.4 Development Based on JDBC

5.4.1 Development Process

Application development based on JDBC covers obtaining the driver JAR package and configuring JDK environment, connecting to a database, running SQL statements, processing result sets, and closing the connection. For details, see [Figure 5-2](#).

Figure 5-2 Application development process based on JDBC



5.4.2 Development Process

5.4.2.1 Obtaining the JAR Package of the Driver and Configuring the JDK Environment

Environment preparation includes obtaining the driver JAR package and configuring the JDK environment.

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-5](#).

Table 5-5 Driver package download list

Version	Download Address
V2.0-3.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

- Verifying the software package integrity on a Linux server:
 - a. Upload the software package and verification package to the same directory on the VM.
 - b. Run the following command to verify the integrity of the software package:


```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

 If **OK** is displayed in the command output, the verification is successful.


```
GaussDB_driver.zip: OK
```
- Verifying the software package integrity on a Windows server:
 - a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
 - b. Run the following command to obtain the hash value of the driver package:


```
certutil -hashfile {local_directory_of_the_driver_package}\{driver_package_name} sha256
```

 - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
 - Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **GaussDB_driver.zip**.
 Example: **certutil -hashfile C:\Users\GaussDB_driver.zip sha256**
 - c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-5**.
 - If they are consistent, the verification is successful.
 - If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Decompressing the Driver Package

Decompress the obtained driver package, find the JDBC driver package **GaussDB-Kernel_Database version number_OS_64bit_Jdbc.tar.gz**, and decompress it. After the decompression, you will obtain the following driver packages in JAR format:

- **opengaussjdbc.jar**: The main class name is **com.huawei.opengauss.jdbc.Driver**. The URL prefix of the database connection is **jdbc:opengauss**. This driver package is recommended. The Java

code examples in this section use the **opengaussjdbc.jar** package by default. This driver package is used when both the PG database and GaussDB are accessed in a JVM process.

- **gsjdbc4.jar**: The main class name is **org.postgresql.Driver**, and the URL prefix of the database connection is **jdbc:postgresql**. This driver package applies to the scenario where services are migrated from the PG database. The driver class and loading path are the same as those before the migration, but the supported APIs are different. The APIs that are not supported need to be adjusted on the service side.
- **gscejdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of database connections is **jdbc:gaussdb**. This driver package contains the dependent libraries related to encryption and decryption that need to be loaded to the encrypted database. This driver package is recommended in encrypted scenarios. Currently, only EulerOS is supported. Before using the **gscejdbc.jar** driver package, set the environment variable **LD_LIBRARY_PATH**. For details, see "Setting Encrypted Equality Queries > Using JDBC to Operate an Encrypted Database" in *Feature Guide*.
- **gsjdbc200.jar**: The main class name is **com.huawei.gauss200.jdbc.Driver**, and the URL prefix of the database connection is **jdbc:gaussdb**. This driver package applies to the scenario where services are migrated from GaussDB 200. The driver class and loading path are the same as those before the migration, but the supported APIs are different. The APIs that are not supported need to be adjusted on the service side.

NOTICE

- The loading paths and URL prefixes of driver classes vary in different driver packages, but their functions are the same.
- The JAR packages of the JDBC release package are classified by architecture. The **gscejdbc.jar** package can be used only when it is consistent with the deployment environment. Other JAR packages do not need to be consistent with the deployment environment.
- The **gsjdbc4** driver package cannot be used to operate the PG database. Although the connection can be successfully established in some versions, some API behaviors are different from those of PG JDBC, which may cause unknown errors.
- The PG driver package cannot be used to operate GaussDB. Although the connection can be successfully established in some versions, some API behaviors are different from those of GaussDB JDBC, which may cause unknown errors.

Configuring the JDK Environment

JDK 1.8 must be configured on the client. JDK supports multiple platforms, such as Windows and Linux. The following uses Windows as an example to describe the configuration method:

- Step 1** Enter the following command in the MS-DOS window (command prompt in Windows) to check the JDK version:

```
java -version
```

Check whether JDK 1.8 has been installed. If JDK is not installed, download the installation package from the [official website](#) and install it.

Step 2 Configure system environment variables.

1. Right-click **My computer** and choose **Properties**.
2. In the **System** window, click **Advanced system settings** in the navigation pane.
3. In the **System Properties** dialog box, click **Environment Variables** on the **Advanced** tab page.
4. In the **System variables** area of the **Environment Variables** dialog box, click **New** or **Edit** to configure system variables. For details about the variables, see [Table 5-6](#).

Table 5-6 Variables

Variable	Operation	Variable Value
JAVA_HOME	<ul style="list-style-type: none">- If the variable exists, click Edit.- If the variable does not exist, click New.	Specifies the Java installation directory. Example: C:\Program Files\Java\jdk1.8.0_131
Path	Click Edit .	<ul style="list-style-type: none">- If <i>JAVA_HOME</i> is configured, add %JAVA_HOME%\bin before the variable value.- If <i>JAVA_HOME</i> is not configured, add the full Java installation path before the variable value: C:\Program Files\Java\jdk1.8.0_131\bin
CLASSPATH	Click New .	%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar

----End

5.4.2.2 Connecting to a Database

After you have obtained the driver JAR package and configured the JDK environment, GaussDB allows you to connect to a database in three modes. This section describes how to connect to a database in the three modes.

5.4.2.2.1 Connection Methods

The database connection method is usually selected based on factors such as security, performance, scalability, and implementation complexity. For details, see [Table 5-7](#).

Table 5-7 Connection Methods

Connection Method	Advantage	Disadvantage	Selection Principle
Non-encrypted	<ul style="list-style-type: none"> • This method is easy to implement and does not require additional encryption and decryption operations. • It features high performance because no encryption or decryption operation is required, which reduces the overhead of data transmission. 	<ul style="list-style-type: none"> • It provides low security and data is easily intercepted or tampered with during transmission. • It is not suitable for sensitive data transmission and may violate privacy protection laws and regulations. 	<p>It applies only to internal networks or environments that have low security requirements.</p>
SSL	<ul style="list-style-type: none"> • It provides encrypted transmission. Data is protected during transmission, ensuring high security. • It supports client/server identity authentication, which enhances security. 	<ul style="list-style-type: none"> • It is complex in configuration and management, and certificates need to be issued and updated. • Some performance overhead may be added, especially when it is used for encrypting and decrypting large amounts of data. 	<p>It applies to environments that require high data transmission security, such as finance and healthcare industries.</p>
UDS	<ul style="list-style-type: none"> • It applies to local communication without passing through the network, featuring a high transmission speed. • No extra network overhead is required, and it is not vulnerable to network attacks. 	<ul style="list-style-type: none"> • It is used only for the communication between processes on the same host but cannot be used for remote connections. 	<p>It applies to local communication scenarios, for example, data exchange between local services and applications.</p>

5.4.2.2.2 Connection Parameter Reference

All connection properties of the **info** parameter are case-sensitive. [Table 5-8](#) describes the common properties.

Table 5-8 Connection properties of the info parameter

Property	Description	Value
PGDBNAME	Specifies the database name. You do not need to set this parameter in the URL because it is automatically parsed from the .properties file.	Property type: string
PGHOST	Specifies the IP address of the host. You do not need to set this parameter in the URL because it is automatically parsed from the .properties file. Both IPv4 and IPv6 addresses are supported. If multiple CNs are configured, use colons (:) to separate their IP addresses and port numbers, and use commas (,) to separate CNs.	Property type: string
PGPORT	Specifies the port number of the host. You do not need to set this parameter in the URL because it is automatically parsed from the .properties file.	Property type: integer
user	Specifies the database user who creates the connection.	Property type: string
password	Specifies the password of the database user.	Property type: string
driverInfo Mode	Controls the output mode of the driver description information.	Property type: string Value range: postgresql or gaussdb . <ul style="list-style-type: none">• postgresql: The driver description related to PostgreSQL is displayed.• gaussdb: The driver description related to GaussDB is displayed. Default value: postgresql

Property	Description	Value
enable_ce	Specifies the encrypted equality query capability.	<p>Property type: integer</p> <p>Value range:</p> <ul style="list-style-type: none"> • 1: JDBC supports the basic capability of encrypted equality query. • Other values: The basic capability of encrypted equality query is not supported. <p>Default value: No default value is provided, indicating that encrypted equality query is not supported.</p>
refreshClientEncryption	Specifies whether the encrypted database supports cache update on the client.	<p>Property type: string</p> <p>Value range:</p> <ul style="list-style-type: none"> • The value 1 or NULL indicates that the encrypted database supports cache update on the client. • Other character strings indicate that the encrypted database does not support cache update on the client. <p>Default value: NULL</p>
loggerLevel	Specifies the logging level.	<p>Property type: string</p> <p>Value range: OFF, INFO, DEBUG, and TRACE (case-insensitive).</p> <ul style="list-style-type: none"> • OFF: The logging function is disabled. • INFO, DEBUG, and TRACE logs record information of different levels. <p>Default value: No default value is provided. If this property is not set, the value INFO is used.</p>

Property	Description	Value
loggerFile	<p>Specifies the log output path (directory and file name). Generally, the log directory and file name must be specified.</p> <p>If only the file name is specified and the directory is not specified, logs are generated in the client running program directory. If no path is configured or the configured path does not exist, log streams are output by default. This parameter has been deprecated and does not take effect. To use this parameter, you can configure it in the java.util.logging .properties file or system properties.</p>	Property type: string
logger	<p>Specifies the log output framework used by the JDBC driver. The JDBC driver supports the log output framework used for interconnecting with user applications. Currently, only the Slf4j-API-based third-party log output framework is supported. For details, see Log Management.</p>	<p>Property type: string</p> <p>Value range: Slf4JLogger and JdkLogger</p> <ul style="list-style-type: none"> • If it is left empty, the JDBC driver uses JdkLogger. • The third-party log framework based on Slf4j-API is used.
allowEncodingChanges	<p>If this parameter is set to true, the character set type can be changed. This parameter is used together with characterEncoding to set the character set. The two parameters are separated by ampersands (&). Example: allowEncodingChanges=true&characterEncoding=UTF8.</p>	<p>Property type: Boolean</p> <p>Value range:</p> <ul style="list-style-type: none"> • true: The character set type can be changed. • false: The character set type cannot be changed. <p>Default value: false</p>

Property	Description	Value
characterEncoding	It must be used together with allowEncodingChanges . When allowEncodingChanges is set to false , characterEncoding can be set only to UTF8 . When allowEncodingChanges is set to true , set characterEncoding as permitted.	Property type: string Value range: UTF8, GBK, Latin1, and GB18030 . NOTE When the database whose character set is GB18030_2022 is connected, setting characterEncoding to GB18030_2022 does not take effect and UTF8 is used by default. You need to set characterEncoding to GB18030 so that the GB18030_2022 characters on the server can be properly parsed. Default value: UTF8
currentSchema	Specifies the schema of the current connection. You need to specify a schema in search-path . If the schema name contains special characters except letters, digits, and underscores (_), you need to enclose the schema name in quotation marks. Note that the schema name is case-sensitive after quotation marks are added. If multiple schemas need to be configured, separate them with commas (,). Example: currentSchema= schema_a," schema-b","schema/c"	Property type: string Default value: If this parameter is not set, the default schema is the username used for the connection.
loadBalanceHosts	Specifies whether to enable the load balancing function.	Property type: Boolean Value range: <ul style="list-style-type: none"> ● true: The shuffle algorithm is used to randomly select a host from the candidates to establish a connection. ● false: Multiple hosts specified in the URL are connected in sequence. Default value: false

Property	Description	Value
<p>autoBalance</p>	<p>Specifies a load balancing policy.</p> <p>NOTICE</p> <ul style="list-style-type: none"> • Load balancing is based on the connection level rather than the transaction level. If the connection is persistent and the load on the connection is unbalanced, the load on the CN may be unbalanced. • This parameter can be used only in distributed scenarios and cannot be used in centralized scenarios. • If the load balancing function is enabled when the DR cluster is connected, query requests are sent to the primary cluster. As a result, the query performance is affected. • When load balancing is enabled, the floating IP address or data IP address can be configured in the URL. If the floating IP address is configured, the system obtains the corresponding data IP address based on the floating IP address and performs load balancing based on the obtained data IP address. Therefore, when configuring the floating IP address or data IP address in the URL, ensure that the network connection of the data IP address is normal. Otherwise, the load balancing function is abnormal. 	<p>Property type: string</p> <p>Value range:</p> <ul style="list-style-type: none"> • If this parameter is set to true, balance, or roundrobin, the JDBC load balancing function is enabled to balance multiple connections of an application to each CN available in the database cluster. <p>Example:</p> <pre>jdbc:opengauss://host1:port1,host2:port2/database?autoBalance=true</pre> <p>JDBC periodically updates the list of available CNs in the entire cluster (data IP addresses of hosts in the CN list). The update interval can be configured using the refreshCNIPListTime parameter. The default value is 10 seconds. For example, the obtained list is as follows: <pre>host1:port1,host2:port2,host3:port3,host4:port4</pre> <p>When load balancing is enabled, host1 and host2 ensure high availability only when they are connected for the first time. Then, the driver selects available CNs from host1, host2, host3, and host4 in sequence to refresh the available CN list. New connections will use the round robin algorithm to select CNs from host1, host2, host3, and host4 for connection.</p> <ul style="list-style-type: none"> • If this parameter is set to priority<i>n</i>, the JDBC-based load balancing function is enabled. Multiple connections of an application are balanced to the first <i>n</i> available CNs configured in the URL. When the first <i>n</i> CNs are unavailable, connections are randomly allocated to other available CNs in the database cluster. <i>n</i> is a positive integer and is less than </p>

Property	Description	Value
		<p>the number of CNs configured in the URL. Example: jdbc:opengauss:// host1:port1,host2:port2,host3:port3,host4: port4/database?autoBalance=priority2</p> <p>JDBC periodically obtains the list of all available CNs in the entire cluster. host1:port1,host2:port2,host3:port3,host4: port4,host5:port5,host6:port6</p> <p>host1 and host2 are in AZ1, and host3 and host4 are in AZ2. The driver preferentially selects host1 and host2 for load balancing. If both host1 and host2 are unavailable, the driver randomly selects a CN from host3, host4, host5, and host6 for connection.</p> <ul style="list-style-type: none"> • If this parameter is set to shuffle, JDBC random load balancing is enabled to randomly and evenly distribute multiple connections of the application to available CNs in the database cluster. Example: jdbc:opengauss:// host1:port1,host2:port2,host3:port3/ database?autoBalance=shuffle <p>JDBC periodically obtains the list of all available CNs in the entire cluster. host1:port1,host2:port2,host3:port3,host4: port4</p> <p>host1, host2, and host3 ensure high availability only when they are connected for the first time. Later, they use the shuffle algorithm to randomly select a CN from the refreshed CN list for connection.</p> <ul style="list-style-type: none"> • If this parameter is set to specified, the JDBC load balancing function is enabled and connections are established only on nodes specified in the user URL. Example:

Property	Description	Value
		<p>jdbc:opengauss:// host1:port1,host2:port2,host3:port3,host4: port4/database? autoBalance=specified&priorityServers=2</p> <p>JDBC identifies host1 and host2 as the nodes for establishing connections in the primary cluster, and identifies host3 and host4 as the nodes for establishing connections in the standby cluster. When a connection is set up, JDBC evenly distributes the connections to host1 and host2. If all the nodes (host1 and host2) in the primary cluster are faulty, JDBC does not attempt to connect to the remaining nodes. After a switchover, JDBC evenly distributes connections to host3 and host4. If all nodes (host3 and host4) in the standby cluster are faulty, JDBC does not attempt to connect to the remaining nodes.</p> <ul style="list-style-type: none"> • If this parameter is set to false, the JDBC load balancing and priority-based load balancing functions are disabled. <p>Default value: false</p>
refreshCNIPListTime	<p>Obtains the cache validity period of the CNs. During connection establishment, the CN status in the database cluster is checked and is reliable within the period specified by refreshCNIPListTime. After the period expires, the status becomes invalid. The IP address list of available CNs is obtained again during the next connection establishment.</p>	<p>Property type: integer Unit: s Value range: 1 to 9999 Default value: 10 Suggestion: Adjust the value based on service requirements and use it together with autoBalance.</p>

Property	Description	Value
hostRecheckSeconds	After JDBC attempts to connect to a host, the host status is saved: connection success or connection failure. This status is trusted within the period specified by hostRecheckSeconds . After the period expires, the status becomes invalid.	Property type: integer Unit: s Value range: 0 to 2147483647. Default value: 10
ssl	Specifies that the database is connected in SSL mode. The SSL options are NonValidatingFactory and certificate.	Property type: Boolean Value range: <ul style="list-style-type: none"> ● true: The database is connected in SSL mode. ● false: The database is not connected in SSL mode. Default value: No default value is provided. If this property is not set, the value false is used.

Property	Description	Value
sslmode	This parameter specifies the SSL authentication mode.	<p>Property type: string</p> <p>Value range: disable, allow, prefer, require, verify-ca, and verify-full.</p> <ul style="list-style-type: none"> ● disable: SSL connection is disabled. ● allow: If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified. ● prefer: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified. ● require: The system only attempts to set up an SSL connection. It neither checks whether the server certificate is issued by a trusted CA, nor checks whether the host name of the server is the same as that in the certificate. ● verify-ca: The system attempts to set up an SSL connection and checks whether the server certificate is issued by a trusted CA. ● verify-full: The system attempts to set up an SSL connection, checks whether the server certificate is issued by a trusted CA, and checks whether the host name of the server is the same as that in the certificate. <p>Default value: No default value is provided. If this property is not set, the value require is used.</p>
sslcert	Specifies the complete path of the certificate file. The certificate type is End Entity .	<p>Property type: string</p> <p>Default value: No default value is provided. If this property is not set, the root directory of the user is read.</p>

Property	Description	Value
sslkey	Specifies the complete path of the key file. You need to convert the key to the DER format before using it. openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt	Property type: string Default value: No default value is provided. If this property is not set, the root directory of the user is read.
sslrootcert	Specifies the name of the SSL root certificate. The root certificate type is CA .	Property type: string Default value: No default value is provided. If this property is not set, the root directory of the user is read.
sslpassword	Specifies the SSL password, which is provided for ConsoleCallbackHandler.	Property type: string
sslpasswordcallback	Specifies the class name of the SSL password provider.	Property type: string Default value: com.huawei.opengauss.jdbc.ssl.jdbcbc4.LibPQFactory.ConsoleCallbackHandler
sslfactory	Specifies the class name used by SSLSocketFactory to establish an SSL connection.	Property type: string Default value: No default value is provided.
sslprivatekeyfactory	Specifies the fully qualified name of the implementation class of the com.huawei.opengauss.jdbc.ssl.PrivateKeyFactory API that implements the private key decryption method. If this parameter is not specified, try the default JDK private key decryption algorithm. If the decryption fails, use com.huawei.opengauss.jdbc.ssl.BouncyCastlePrivateKeyFactory . You need to provide the bcpkix-jdk15on.jar package. The recommended version is 1.65 or later.	Property type: string Default value: No default value is provided. If this property is not set, the system attempts to use the default JDK private key decryption algorithm for decryption.
sslfactoryarg	The value is an optional parameter of the constructor function of the sslfactory class. (This parameter is not recommended.)	Property type: string Default value: No default value is provided.

Property	Description	Value
sslhostnamerverifier	Specifies the class name of the host name verifier. The API must implement javax.net.ssl.HostnameVerifier . The default value is org.postgresql.ssl.PGjdbcHostnameVerifier .	Property type: string Default value: No default value is provided. If this property is not set, org.postgresql.ssl.PGjdbcHostnameVerifier is used.

Property	Description	Value
loginTimeout	<p>Specifies the waiting time for establishing the database connection. When multiple IP addresses are configured in the URL, if the time for obtaining the connection exceeds the value of this parameter, the connection fails and the subsequent IP addresses are not tried.</p> <p>The value of loginTimeout is related to connectTimeout and socketTimeoutInConnecting. The calculation formula is as follows: loginTimeout = (connectTimeout + Connection authentication time + Initialization statement execution time) x Number of nodes.</p>	<p>Property type: integer</p> <p>Unit: s</p> <p>Value range: 0 to 2147483647. The value 0 indicates that the timeout mechanism does not take effect.</p> <p>Default value: 0</p> <p>Suggestion: After this parameter is set, an asynchronous thread is started each time a connection is established. If there are a large number of connections, the pressure on the client may increase. You are advised to set it as follows: max(connectTimeout, socketTimeoutInConnecting) x Number of nodes. This prevents connection failures when the network is abnormal and the <i>n</i>th IP address is the IP address of the primary node. You can adjust it based on the actual service situation.</p>

Property	Description	Value
	<p>NOTICE</p> <ul style="list-style-type: none"> This parameter sets the time for attempting to connect to all IP addresses in a list. If this parameter is set to a small value, the subsequent IP addresses in the list may fail to be connected. For example, if three IP addresses are set, loginTimeout is set to 5s, and it takes 5s to connect to the first two IP addresses, the third IP address cannot be connected. When any of the CPU, memory, and I/O load approaches 100%, the connection is slow, which may cause connection timeout. You can locate the fault as follows: <ul style="list-style-type: none"> Log in to a physical machine with slow connections or use a management tool to query the resource load. You can run the top command to check the CPU usage, run the free command to check the memory usage, and run the iostat command to check the I/O load. In addition, you can check the monitoring logs in the CM Agent and the monitoring records on the database O&M platform. For peak load scenarios caused by a large number of slow queries in a short period of time, you can use the port specified by [<i>Port number of the database server</i> + 1] to query the <code>pg_stat_activity</code> view. For slow queries, you can use the system function pg_terminate_backend(pid int) to kill sessions. If service overloading exists for a long time (that is, there is no obvious slow query, or new queries still become slow after slow queries are killed), reduce the service load and increase database resources. 	

Property	Description	Value
connectTimeout	Specifies the timeout interval for connecting to a server OS. If the time taken to connect to a server OS exceeds the value specified, the connection is interrupted. When multiple IP addresses are configured in the URL, this parameter indicates the timeout interval for connecting to a single IP address.	Property type: integer Unit: s Value range: 0 to 2147483647. The value 0 indicates that the timeout mechanism does not take effect. Default value: 0
socketTimeout	Specifies the timeout period for a socket read operation. If the time taken to read data streams from a server exceeds the value specified, the connection is closed. If this parameter is not set, the client waits for a long time when the database process is abnormal. You are advised to set this parameter based on the SQL execution time acceptable to services.	Property type: integer Unit: s Value range: 0 to 2147483647. The value 0 indicates that the timeout mechanism does not take effect. Default value: 0
socketTimeoutInConnecting	This parameter specifies the timeout value of the socket read operation during the connection establishment. If the time of reading data streams from the server exceeds the threshold, it attempts to search for the next node for connection.	Property type: integer Unit: s Value range: 0 to 2147483647. The value 0 indicates that the timeout mechanism does not take effect. Default value: 5
statementTimeout	Specifies the timeout interval for executing a statement in a connection. If the execution time of a statement exceeds this value, the statement execution is canceled.	Property type: integer Unit: ms Value range: 0 to 2147483647. The value 0 indicates that the timeout mechanism does not take effect. Default value: 0

Property	Description	Value
cancelSignalTimeout	Controls "connect timeout" and "socket timeout" of a cancel command, which may cause blocking. If the cancel command does not respond within the specified time, the connection is interrupted to reduce the occupation of client resources.	Property type: integer Unit: s Value range: 0 to 2147483647. The value 0 indicates that the timeout mechanism does not take effect. Default value: 10
tcpKeepAlive	Specifies whether to enable TCP keepalive detection.	Property type: Boolean Value range: <ul style="list-style-type: none"> • true: TCP keepalive detection is enabled. • false: TCP keepalive detection is disabled. Default value: false
logUnclosedConnections	The client may leak a connection object because it does not call the close() method to close the connection object. Such object will be recycled and finalized using the finalize() method.	Property type: Boolean Value range: <ul style="list-style-type: none"> • true: If the close() method is not called, the finalize() method will be used to close the corresponding Connection object. • false: If the close() method is not called, the corresponding Connection object will not be closed. Default value: false
assumeMinServerVersion	Specifies the version of the server to be connected. If the value of assumeMinServerVersion is greater than or equal to 9.0 , the number of packets to be sent are reduced during connection establishment. The client does not send a request to set the floating point to 3 (that is, retain the original floating point 2).	Property type: string Default value: No default value is provided.

Property	Description	Value
ApplicationName	Specifies the name of the JDBC driver that is being connected. You can query the PGXC_STAT_ACTIVITY system view on CN to view the information about the client that is being connected. The application_name field indicates the JDBC driver name.	Property type: string Default value: PostgreSQL JDBC Driver
connectionExtraInfo	Specifies whether the driver reports the driver deployment path, process owner, and URL connection configuration information to the database.	Property type: Boolean Value range: <ul style="list-style-type: none"> • true: The JDBC driver reports the driver deployment path, process owner, and URL connection configuration information to the database and displays the information in the connection_info parameter. In this case, you can query the information from PG_STAT_ACTIVITY or PGXC_STAT_ACTIVITY. • false: The driver does not report the driver deployment path, process owner, and URL connection configuration information to the database. Default value: false
autosave	Specifies the action that the driver should perform upon a query failure.	Property type: string Value range: always , never , and conservative . <ul style="list-style-type: none"> • always: The JDBC driver sets a savepoint before each query and rolls back to the savepoint if the query fails. • never: There is no savepoint. • conservative: A savepoint is set for each query. However, the system rolls back and retries only when there is an invalid statement. Default value: never

Property	Description	Value
protocolVersion	Specifies the connection protocol version.	Property type: integer Value range: 1 and 3 . <ul style="list-style-type: none"> • 1: The V1 server is connected. • 3: The V5 server is connected. Default value: No default value is provided. If this property is not set, the value 3 is used.
prepareThreshold	Specifies the number of times that the PreparedStatement object is executed before the parsed statement on the server is used. NOTE You are advised not to use JDBC and preparedStatement to execute password-related statements (for example, CREATE USER user_name WITH PASSWORD '*****'). This is because when the number of execution times reaches the value specified by prepareThreshold , the database caches SQL statements but does not cache the passwords for security purposes. When the preparedStatement is executed again, error message "Password must contain at least 8 characters" will be displayed.	Property type: integer Value range: 0 to 2147483647. Default value: 5 . The default value indicates that when the same PreparedStatement object is executed for five or more times, the parse message is not sent to the server to parse the statement. Instead, the statement that has been parsed on the server is used.
preparedStatementCacheQueries	Specifies the maximum number of queries generated by caching PreparedStatement objects in each connection.	Property type: integer Value range: 0 to 2147483647. The value 0 indicates that the cache function is disabled. Default value: 256 . If more than 256 different queries are used in the preparedStatement() call, the least recently used query cache will be discarded. Suggestion: Set the value based on service requirements. This parameter is used together with prepareThreshold .

Property	Description	Value
preparedStatementCacheSizeMiB	Specifies the maximum size of queries generated by caching PreparedStatement objects in each connection.	Property type: integer Unit: MB Value range: 0 to 2147483647. The value 0 indicates that the cache function is disabled. Default value: 5 . If the size of the cached queries exceeds 5 MB, the least recently used query cache will be discarded.
databaseMetadataCacheFields	Specifies the maximum number of columns that can be cached in each connection.	Property type: integer Value range: 0 to 2147483647. The value 0 indicates that the cache function is disabled. Default value: 65536
databaseMetadataCacheFieldsMiB	Specifies the maximum size of columns that can be cached in each connection.	Property type: integer Unit: MB Value range: 0 to 2147483647. The value 0 indicates that the cache function is disabled. Default value: 5
stringtype	Specifies the type of the parameter transferred to the database when the java.sql.PreparedStatement#setString method is called.	Property type: string Value range: unspecified and varchar . <ul style="list-style-type: none"> ● varchar: Parameters are sent to the server as varchar parameters. ● unspecified: Parameters are sent to the server without their types specified, and the server will attempt to infer the appropriate types. Default value: varchar

Property	Description	Value
batchMode	<p>Specifies whether to use the batch mode.</p> <p>NOTE If batchMode is set to on, the data type of each column is the same as that specified by the first data record. If the data types are mixed, an error may be reported or the inserted data may be abnormal.</p>	<p>Property type: string</p> <p>Value range:</p> <ul style="list-style-type: none"> • on: The batch mode is enabled to improve the batch update performance. If batchMode is set to on, the returned result is [count, 0, 0...0]. The first element in the array is the total number of records affected in batches. • off: The batch mode is disabled. If batchMode is set to off, the returned result is [1, 1, 1...1]. Each element in the array corresponds to the number of affected records in a single modification. <p>Default value: on</p> <p>Suggestion: If the service framework (such as hibernate) checks the return value during batch update, you can set this parameter to solve the problem.</p>
fetchsize	<p>Specifies the default fetchsize for statements in the created connection.</p> <p>Specifies the number of rows fetched from ResultSet each time. It has the same function as defaultRowFetchSize. If fetchsize and defaultRowFetchSize are set at the same time, fetchsize prevails.</p>	<p>Property type: integer</p> <p>Value range: 0 to 2147483647.</p> <p>Default value: 0. It indicates that all results are obtained from the database at a time.</p> <p>Suggestion: You are advised to set this parameter based on the amount of data queried by services and the memory of the client. When setting fetchsize, disable automatic commit (set autocommit to false). Otherwise, the setting of fetchsize does not take effect.</p>
defaultRowFetchSize	<p>Specifies the number of rows fetched from ResultSet each time. Limiting the number of rows read each time in a database access request can avoid unnecessary memory consumption, thereby avoiding the out of memory exception.</p>	<p>Property type: integer</p> <p>Value range: 0 to 2147483647.</p> <p>Default value: 0. It indicates that all results are obtained from the database at a time.</p>

Property	Description	Value
reWriteBatchedInserts	Specifies whether to rewrite SQL statements during batch insertion. When this parameter is used, batchMode must be set to off .	Property type: Boolean Value range: <ul style="list-style-type: none"> • true: <i>N</i> insert statements can be combined into one: INSERT INTO TABLE_NAME VALUES (values1, ..., valuesN), ..., (values1, ..., valuesN) • false: SQL statements are not rewritten during batch insertion. Default value: false
unknownLength	Specifies the length of some GaussDB types (such as TEXT) whose length is unknown and not defined when they are returned by functions such as ResultSetMetaData.getColumnDisplaySize and ResultSetMetaData.getPrecision.	Property type: integer Value range: 0 to 2147483647. Default value: 2147483647
binaryTransfer	Specifies whether data is sent and received in binary format.	Property type: Boolean Value range: <ul style="list-style-type: none"> • true: enabled. • false: disabled. Default value: false
binaryTransferEnable	Specifies types for which binary transmission is enabled. Types are separated by commas (,), for example, binaryTransferEnable=Integer4_ARRAY,Integer8_ARRAY . Select either the name or OID. For example, if the name is BLOB and the OID is 88, set binaryTransferEnable to BLOB or 88 .	Property type: string Default value: No default value is provided.

Property	Description	Value
binaryTransferDisable	Specifies types for which binary transmission is disabled. Use commas (,) to separate names or OIDs. If binaryTransferDisable and binaryTransferEnable have the same value, the disabling function is used.	Property type: string Default value: No default value is provided.
blobMode	Sets the setBinaryStream() method to assign values to different types of data. You are advised to set this parameter to on for systems migrated from ORA and MySQL and to off for systems migrated from PG.	Property type: string Value range: <ul style="list-style-type: none"> • on: Values are assigned to BLOB data. • off: Values are assigned to bytea data. Default value: on
socketFactory	Specifies the name of the class used to create a socket connection with the server. This class must implement the javax.net.SocketFactory API and define a constructor with no parameter or a single string parameter.	Property type: string
socketFactoryArg	The value is an optional parameter of the constructor function of the socketFactory class and is not recommended.	Property type: string
receiveBufferSize	Sets SO_RCVBUF on a connection stream.	Property type: integer Unit: byte Value range: -1 to 2147483647 Default value: -1 . It indicates that the buffer size is not set.
sendBufferSize	Sets SO_SNDBUF on a connection stream.	Property type: integer Unit: byte Value range: -1 to 2147483647 Default value: -1 . It indicates that the buffer size is not set.

Property	Description	Value
preferQueryMode	Specifies the query mode.	Property type: string Value range: simple , extended , extendedForPrepared , and extendedCacheEverything . <ul style="list-style-type: none"> ● In simple mode, only Q messages are sent. Only text is supported. The parse and bind operations are not supported. ● In extended mode, parse, bind, and execute operations are supported. ● In extendedForPrepared mode, extended query applies to only the Prepared Statement objects, and the Statement objects support only simple query. ● In extendedCacheEverything mode, the query generated by each Statement object is cached. Default value: extended
ApplicationType	Sets whether to enable distributed write and query. This parameter is valid only when the database is in the GTM-free mode.	Property type: string Value range: not_perfect_sharding_type and perfect_sharding_type <ul style="list-style-type: none"> ● not_perfect_sharding_type: enables distributed write and query. ● perfect_sharding_type: disables distributed write and query by default. Distributed write and query can be performed only after <code>/* multinode */</code> is added to the SQL statement. Default value: not_perfect_sharding_type

Property	Description	Value
priorityServers	<p>Specifies that the first <i>n</i> nodes configured in the URL are preferentially connected as the primary cluster. It is used in streaming DR scenarios.</p> <p>Example: jdbc:postgresql://host1:port1,host2:port2,host3:port3,host4:port4/database?priorityServers=2. That is, host1 and host2 are primary cluster nodes, and host3 and host4 are standby cluster nodes for DR.</p>	<p>Property type: integer</p> <p>Value range: a number greater than 0 and less than the number of CNs configured in the URL.</p> <p>Default value: NULL</p>
usingEip	<p>Specifies whether to use an elastic IP address to implement load balancing.</p> <p>NOTICE If usingEip is set to true, the hosts in the URL must use elastic IP addresses. If usingEip is set to false, the hosts in the URL must use data IP addresses. Otherwise, the priority/<i>n</i> load balancing policy will become invalid.</p>	<p>Property type: Boolean</p> <p>Value range:</p> <ul style="list-style-type: none"> ● true : Elastic IP addresses are used to implement load balancing. ● false: Data IP addresses are used to implement load balancing. <p>Default value: true</p>
tracerInterfaceClass	<p>Specifies the fully qualified name of the implementation class of the org.postgresql.log.Tracer API that implements the method for obtaining traceld.</p>	<p>Property type: string</p> <p>Default value: NULL</p>
use_boolean	<p>Sets the OID type bound to the setBoolean() method in extended mode.</p>	<p>Property type: Boolean</p> <p>Value range:</p> <ul style="list-style-type: none"> ● false: The int2 type is bound. ● true: The Boolean type is bound. <p>Default value: false</p>

Property	Description	Value
allowRead Only	Specifies whether to enable the read-only mode for a connection.	Property type: Boolean Value range: <ul style="list-style-type: none"> ● true: The read-only mode is enabled. ● false: The read-only mode is disabled. In this case, calling <code>connection.setReadOnly(true)</code> does not take effect, and data can still be modified. Default value: true
TLSCiphers Supported	Specifies the supported TLS cipher suite.	Property type: string Default value: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
stripTrailingZeros	This parameter specifies whether to remove trailing zeros after the decimal point of the numeric type. It is valid only for <code>ResultSet.getObject(int columnIndex)</code> .	Property type: Boolean Value range: <ul style="list-style-type: none"> ● true: Trailing zeros after the decimal point of the numeric type are removed. ● false: Trailing zeros after the decimal point of the numeric type are not removed. Default value: false
enableTimeZone	Specifies whether to enable the time zone setting on the client.	Property type: Boolean Value range: <ul style="list-style-type: none"> ● true: The time zone setting on the client is enabled. The JVM time zone is obtained to specify the database time zone. ● false: The time zone setting on the client is disabled. Instead, the database time zone is used. Default value: true

Property	Description	Value
enableStandbyRead	Specifies whether to enable the read mode of the standby node.	Property type: Boolean Value range: <ul style="list-style-type: none"> ● true: enabled. ● false: disabled. Default value: false
oracleCompatible	This is used to set the ORA-compatible features of driver APIs.	Property type: string Value range: <ul style="list-style-type: none"> ● true: enables all ORA compatibility features on the driver side. ● false: disables all ORA compatibility features on the driver side. ● tag1,tag2,tag3: configures one or more tags. Use commas (,) to separate tags, indicating that the ORA compatibility features on the driver side are enabled. Each tag corresponds to an ORA compatibility feature. Currently, the following tags are supported: <ul style="list-style-type: none"> - getProcedureColumns: The behavior of the DatabaseMetaData#getProcedureColumns API is compatible with that in ORA. - batchInsertAffectedRows: After reWriteBatchedInserts is enabled, the result returned by the Statement#executeBatch API is compatible with that in ORA. Default value: false
printSqlInLog	Specifies whether to output SQL statements in exception information or logs.	Property type: Boolean Value range: <ul style="list-style-type: none"> ● true: enabled. ● false: disabled. Default value: true

Property	Description	Value
setFloat	Specifies whether the OID transferred to the kernel is an OID of float4 when setFloat is called or setObject is set to float .	Property type: Boolean Value range: true : The OID transferred to the kernel is of the float4 type. false : The OID transferred to the kernel is of the float8 type. Default value: false

The JDBC driver can run properly after DDL operations are performed to change the table structure.

jdbc.version_num is used only to transfer the JDBC version number to the kernel during connection establishment. Do not use it for other purposes.

jdbc.version_num is supported in 503.1 and later versions. (The **jdbc.version** parameter was used to implement this function and will be deprecated in later versions.) The usage rules of **jdbc.version_num** are as follows:

jdbc.version_num	Version	Range	Usage Rule	Kernel Judgment Rule	Remarks
502xx	505.2	50200-50299	The minimum available value must be used in the version.	Version number ≥ 502xx (latest value)	Related compatibility changes must be incorporated into branches of later versions.
501xx	505.1	50100-50199	The minimum available value must be used in the version.	Version number ≥ 501xx (latest value)	Related compatibility changes must be incorporated into branches of later versions.

jdbc.version_num	Version	Range	Usage Rule	Kernel Judgment Rule	Remarks
500xx	505.0	50000-50099	The minimum available value must be used in the version.	Version number \geq 500xx (latest value)	Related compatibility changes must be incorporated into branches of later versions.
301xx	503.1	30100-30199	The minimum available value must be used in the version.	Version number \geq 301xx (latest value)	Related compatibility changes must be incorporated into branches of later versions.
300xx	503.0	30000-30099	The minimum available value must be used in the version.	Version number \geq 300xx (latest value)	Related compatibility changes must be incorporated into branches of later versions.

5.4.2.2.3 Connecting to a Database in Non-Encrypted Mode

To connect to a database in non-encrypted mode, you need to load a driver and then establish a database connection. This section describes methods of loading drivers, APIs for creating database connections, and APIs for non-encrypted connections.

Methods of Loading Drivers

You can use either of the following methods to load the driver (**opengaussjdbc.jar** is used as an example):

- Load a driver at any position in code before creating a connection.

```
Class.forName("com.huawei.opengauss.jdbc.Driver");
```

- Specify the driver name when a JVM is started. This parameter is used to execute Java code in the DOS window or on Linux. **jdbctest** is the name of a test application.

```
java -Djdbc.drivers=com.huawei.opengauss.jdbc.Driver jdbctest;
```

NOTE

- If **gsjdbc4.jar** is used, change the driver class name to **org.postgresql.Driver**.
- GaussDB is compatible with PG in the use of JDBC. Therefore, when two JDBC drivers are used in the same process, class names may conflict.
- JDBC of this version does not support identity & access management suite (IAM) for authentication.
- Compared with the PG driver, the GaussDB JDBC driver has the following enhanced features:
 - SHA-256 is supported for encrypted login.
 - The third-party log framework that implements the sf4j API can be connected.
 - Distributed load balancing at the connection level is supported.
 - DR failover is supported.

APIs for Creating Database Connections

JDBC provides three APIs for creating database connections. For details about the **url**, **info**, **user**, and **password** parameters, see [Table 5-9](#).

API 1: `DriverManager.getConnection(String url)`. You need to write a database username and password in a URL, which is insecure and not recommended.

API 2: `DriverManager.getConnection(String url, String user, String password)`. For details, see [Connecting to a Database Using API 2](#).

API 3: `DriverManager.getConnection(String url, Properties info)`. For details, see [Connecting to a Database Using API 3](#).

Table 5-9 Database connection parameters

Parameter	Description
url	<p>Database connection descriptor when opengaussjdbc.jar is used. The format is as follows:</p> <ul style="list-style-type: none"> • jdbc:opengauss: • jdbc:opengauss:database • jdbc:opengauss://host/database • jdbc:opengauss://host:port/database • jdbc:opengauss://host:port/database?param1=value1&param2=value2 • jdbc:opengauss://host1:port1,host2:port2/database?param1=value1&param2=value2 <p>NOTE</p> <ul style="list-style-type: none"> • If gsjdbc200.jar is used, replace jdbc:opengauss with jdbc:gaussdb. • database indicates the name of the database to connect. The database name is the same as the username by default. • host indicates the name or IP address of the database server. For security reasons, the database CN prevents unauthorized access from other nodes within the cluster. To access the CN from inside the cluster, deploy the JDBC program on the host where the CN is located and set host to 127.0.0.1. Otherwise, the error message "FATAL: Forbid remote connection with trust method!" may be displayed. It is recommended that the service system be deployed outside the cluster. Otherwise, the database performance may be affected. By default, the localhost is used to connect to the server. • port indicates the port number of the database server. By default, the database on port 5431 of the localhost is connected. • param indicates a database connection attribute. Parameters can be configured in the URL. The URL starts with a question mark (?), uses an equal sign (=) to assign values to parameters, and uses ampersands (&) to separate parameters. • value indicates the database connection attribute values. • The connectTimeout and socketTimeout parameters must be set for connection. If they are not set, the default value 0 is used, indicating that the connection will not time out. When the network between a DN and the client is faulty, the client cannot receive the ACK packet from the DN and retries transmission repeatedly. A timeout error is reported only when the number of retransmission times reaches 15 (the default value). As a result, the RTO is high. • You are advised to ensure the validity of the URL when using the standard JDBC API to establish a connection. Otherwise, connection may fail. If the error information contains the original URL character string, it may cause sensitive information leakage. • Sensitive information such as passwords and credentials cannot be configured in URLs. You are advised to configure sensitive information using connection properties. For details, see Connecting to a Database Using API 3.
info	<p>Database connection property. For details about all parameters, see Connection Parameter Reference.</p>

Parameter	Description
user	Database user.
password	Password of the database user.

Connecting to a Database Using API 2

The following uses **opengaussjdbc.jar** as an example to describe how to use the `DriverManager.getConnection(String url, String user, String password)` API to establish a database connection. The procedure is as follows:

Step 1 Import `java.sql.Connection` and `java.sql.DriverManager`.

`java.sql.Connection` is a database connection API. The `getConnection()` method of `java.sql.DriverManager` is used to connect applications to a database. In addition, you need to import other APIs and classes based on the actual application scenario. For details, see [JDBC Interface Reference](#).

```
import java.sql.Connection;  
import java.sql.DriverManager;
```

Step 2 Specify the database **sourceURL** (change *\$ip*, *\$port*, and *database* as required), username, and password.

Writing the username and password to code has great security risks. You are advised to store the username and password in environment variables.

```
String sourceURL = "jdbc:opengauss://$ip:$port/$database";  
String userName = System.getenv("EXAMPLE_USERNAME_ENV");  
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

Step 3 Load the driver.

1. Add the **opengaussjdbc.jar** package to the code running tool (such as IDE).
2. Load the database driver **com.huawei.opengauss.jdbc.Driver** as follows:

```
String driver = "com.huawei.opengauss.jdbc.Driver";  
Class.forName(driver);
```

Step 4 Establish a database connection.

Call `DriverManager.getConnection(String url, String user, String password)` to connect to the database.

```
Connection conn = DriverManager.getConnection(sourceURL, userName, password);
```

----End

Connecting to a Database Using API 3

The following uses **opengaussjdbc.jar** as an example to describe how to use the `DriverManager.getConnection(String url, Properties info)` API to establish a database connection. The procedure is as follows:

Step 1 Import `java.sql.Connection`, `java.sql.DriverManager`, and `java.util.Properties`.

The `setProperty()` method of `java.util.Properties` is used to set attribute values of the `Properties` object. In addition, you need to import other APIs and classes based on the actual application scenario. For details, see [JDBC Interface Reference](#).

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.util.Properties;
```

Step 2 Specify the database **sourceURL** (change *\$ip*, *\$port*, and *database* as required), username, and password.

Writing the username and password to code has great security risks. You are advised to store the username and password in environment variables.

```
String sourceURL = "jdbc:opengauss://$ip:$port/$database";  
String userName = System.getenv("EXAMPLE_USERNAME_ENV");  
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

Step 3 Create a `Properties` object and set **userName** and **password** as the attribute values of the object.

```
Properties info = new Properties();  
info.setProperty("user", userName);  
info.setProperty("password", password);
```

Step 4 Load the driver.

1. Add the **opengaussjdbc.jar** package to the code running tool (such as IDE).
2. Load the database driver **com.huawei.opengauss.jdbc.Driver** as follows:

```
String driver = "com.huawei.opengauss.jdbc.Driver";  
Class.forName(driver);
```

Step 5 Establish a database connection.

Call `DriverManager.getConnection(String url, Properties info)` to connect to the database.

```
Connection conn = DriverManager.getConnection(sourceURL, info);
```

----End

5.4.2.2.4 Connecting to a Database in SSL Mode

When establishing connections to the GaussDB server using JDBC, you can enable SSL connections to encrypt client and server communications for security of sensitive data transmission on the Internet. You can use the `NonValidatingFactory` channel or certificate authentication to connect to the database in SSL mode. In certificate-based authentication mode, a client and a server authenticate each other. In this section, the `DriverManager.getConnection(String url, Properties info)` API is used to connect to a database.

Method 1: NonValidatingFactory Channel

Prerequisites: You have obtained the certificates and private key file required by the server and configured the server.

NOTE

For details about how to generate and obtain a certificate, contact an administrator. For details about how to configure the certificate on the server, contact an administrator.

The following uses **opengaussjdbc.jar** as an example to describe how to connect to a database through the `NonValidatingFactory` channel as follows:

Step 1 Import `java.sql.Connection`, `java.sql.DriverManager`, and `java.util.Properties`.

In addition, you need to import other APIs and classes based on the actual application scenario. For details, see [JDBC Interface Reference](#).

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

Step 2 Specify the database **sourceURL** (change `$ip`, `$port`, and `database` as required), username, and password.

Writing the username and password to code has great security risks. You are advised to store the username and password in environment variables.

```
String sourceURL = "jdbc:opengauss://$ip:$port/database";
Properties urlProps = new Properties();
urlProps.setProperty("user", System.getenv("EXAMPLE_USERNAME_ENV"));
urlProps.setProperty("password", System.getenv("EXAMPLE_PASSWORD_ENV"));
```

Step 3 Set the SSL attribute to **true** to use the `NonValidatingFactory` channel.

```
urlProps.setProperty("ssl", "true");
urlProps.setProperty("sslfactory", "com.huawei.opengauss.jdbc.ssl.NonValidatingFactory");
```

Step 4 Load the driver.

1. Add the **opengaussjdbc.jar** package to the code running tool (such as IDE).
2. Load the database driver **com.huawei.opengauss.jdbc.Driver** as follows:

```
Class.forName("com.huawei.opengauss.jdbc.Driver");
```

Step 5 Establish a database connection.

Call `DriverManager.getConnection(String url, Properties info)` to connect to the database.

```
Connection conn = DriverManager.getConnection(sourceURL,urlProps);
```

----End

Method 2: Certificate-based Authentication

Prerequisites: You have obtained the certificates and private key file required by the server and configured the server. You have obtained the **client.crt** client certificate, **ca.cert.pem** root certificate, and **client.key.pk8** client private key file required by the client. [Step 3](#) describes how to configure the certificates and private key file on the client.

NOTE

For details about how to generate and obtain a certificate, contact an administrator. For details about how to configure the certificate on the server, contact an administrator.

The following uses **opengaussjdbc.jar** as an example to describe how to configure certificates on the client to connect to a database as follows:

Step 1 Import `java.sql.Connection`, `java.sql.DriverManager`, and `java.util.Properties`.

In addition, you need to import other APIs and classes based on the actual application scenario. For details, see [JDBC Interface Reference](#).

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

Step 2 Specify the database **sourceURL** (change *\$ip*, *\$port*, and *database* as required), username, and password.

Writing the username and password to code has great security risks. You are advised to store the username and password in environment variables.

```
String sourceURL = "jdbc:opengauss://$ip.$port/database";
Properties urlProps = new Properties();
urlProps.setProperty("user", System.getenv("EXAMPLE_USERNAME_ENV"));
urlProps.setProperty("password", System.getenv("EXAMPLE_PASSWORD_ENV"));
```

Step 3 Set the SSL attribute to **true** and configure the **client.crt** client certificate, **cacert.pem** root certificate, and **client.key.pk8** client private key on the client.

```
urlProps.setProperty("ssl", "true");
urlProps.setProperty("sslcert", "client.crt");
urlProps.setProperty("sslrootcert", "cacert.pem");
urlProps.setProperty("sslkey", "client.key.pk8");
```

NOTE

Before using the client private key file, convert **client.key** to **client.key.pk8**.

```
* openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt
* openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1 PBE-MD5-DES
* openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1 PBE-SHA1-3DES
```

The preceding algorithms are not recommended because they are not that secure.

If the customer needs to use a higher-level private key encryption algorithm, the following private key encryption algorithms can be used after the **bouncycastle** or a third-party private key is used to decrypt the password package:

```
* openssl pkcs8 -in client.key -topk8 -outform DER -out client.key.der -v2 AES128
* openssl pkcs8 -in client.key -topk8 -outform DER -out client.key.der -v2 aes-256-cbc -iter 1000000
* openssl pkcs8 -in client.key -topk8 -out client.key.der -outform Der -v2 aes-256-cbc -v2prf
hmacWithSHA512
* Enable BouncyCastle: Introduce the bcpkix-jdk15on.jar and bcprov-ext-jdk15on.jar packages for
projects that use JDBC. The recommended version is 1.65 or later.
```

Step 4 Configure **sslmode**.

Set **sslmode** to **require**, **verify-ca**, or **verify-full**. For details about the parameters, see **sslmode**. You can select one of them based on the application scenario.

```
/* Set sslmode to require. */
urlProps.setProperty("sslmode", "require");
/* Set sslmode to verify-ca. */
urlProps.setProperty("sslmode", "verify-ca");
/* Set sslmode to verify-full (verification in Linux). */
urlProps.setProperty("sslmode", "verify-full");
```

Step 5 Load the driver.

1. Add the **opengaussjdbc.jar** package to the code running tool (such as IDE).
2. Load the database driver **com.huawei.opengauss.jdbc.Driver** as follows:

```
Class.forName("com.huawei.opengauss.jdbc.Driver");
```

Step 6 Establish a database connection.

Call **DriverManager.getConnection(String url, Properties info)** to connect to the database.

```
Connection conn = DriverManager.getConnection(sourceURL,urlProps);
```

----End

5.4.2.2.5 Connecting to a Database in UDS Mode

The UDS is used for data exchange between different processes on the same host. You can add **unixsocket** to obtain the socket factory.

Prerequisites: You have referenced **unixsocket-core-XXX.jar**, **unixsocket-common-XXX.jar**, and **unixsocket-native-common-XXX.jar**. XXX indicates the version number. The version numbers of the referenced JAR packages must be consistent.

The following uses **opengaussjdbc.jar** as an example to describe how to connect to a database through a UDS as follows:

Step 1 Import java.sql.Connection, java.sql.DriverManager, and java.util.Properties.

In addition, you need to import other APIs and classes based on the actual application scenario. For details, see [JDBC Interface Reference](#).

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

Step 2 Specify the username and password of the database.

Writing the username and password to code has great security risks. You are advised to store the username and password in environment variables. Set the username and password to the property values of the Properties object.

```
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Properties properties = new Properties();
properties.setProperty("user", userName);
properties.setProperty("password", password);
```

Step 3 Load the driver.

1. Add the **opengaussjdbc.jar** package to the code running tool (such as IDE).
2. Load the database driver **com.huawei.opengauss.jdbc.Driver** as follows:

```
String driver = "com.huawei.opengauss.jdbc.Driver";
Class.forName(driver);
```

Step 4 Specify the database *\$ip*, *\$port*, *database*, **socketFactory**, and **socketFactoryArg**.

Change *\$ip* and *\$port* as required. Set *database* to **localhost**, **socketFactory** to **org.newsclub.net.unix.AFUNIXSocketFactory\$FactoryArg**, and **socketFactoryArg** to *[path-to-the-unix-socket]* to connect to the database in UDS mode. For details about the **socketFactory** and **socketFactoryArg** parameters, see [socketFactory](#) and [socketFactoryArg](#).

```
Connection conn = DriverManager.getConnection("jdbc:opengauss://$ip:$port/localhost?
socketFactory=org.newsclub.net.unix" +
".AFUNIXSocketFactory$FactoryArg&socketFactoryArg=[path-to-the-unix-socket]",properties);
System.out.println("Connection Successful!");
```

NOTE

Set the **socketFactoryArg** parameter based on the actual path. The value must be the same as that of the GUC parameter **unix_socket_directory**.

----End

5.4.2.3 Running SQL Statements

This section shows you the processes of creating the **customer_t1** table by [Running a Common SQL Statement](#), inserting data in batches by [Executing a](#)

[Prepared Statement for Insertion](#), update data by [Executing a Prepared Statement for Update](#), and [Creating and Calling a Stored Procedure](#).

Running a Common SQL Statement

SQL statements are run on applications to operate a database. Operations such as SELECT, UPDATE, INSERT, and DELETE can be performed on XML data.

The prerequisite is that you have connected to the database using the connection object conn. Create the **customer_t1** table as follows:

- Step 1** Create a statement object by calling the createStatement method of the Connection API.

```
Statement stmt = conn.createStatement();
```

- Step 2** Run the SQL statement by calling the executeUpdate method of the Statement API.

```
int rc = stmt.executeUpdate("CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name VARCHAR(32));");
```

- Step 3** Close the statement object stmt by calling the close method of the Statement API.

```
stmt.close();
```

----End

NOTE

- If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. As VACCUUM is not supported in a transaction block, if one of the statements fails, the entire request will be rolled back.
- Use semicolons (;) to separate statements. Stored procedures, functions, and anonymous blocks do not support multi-statement execution. When **preferQueryMode** is set to **simple**, the statement does not execute the parsing logic, and the semicolons (;) cannot be used to separate statements in this scenario.
- The slash (/) can be used as the terminator for creating a single stored procedure, function, anonymous block, or package body. When **preferQueryMode** is set to **simple**, the statement does not execute the parsing logic, and the slash (/) cannot be used as the terminator in this scenario.
- JDBC caches SQL statements in prepareStatement, which may cause memory bloat. If the JVM memory is small, you are advised to adjust **preparedStatementCacheSizeMiB** or **preparedStatementCacheQueries**.

Executing a Prepared Statement for Insertion

When a prepared statement processes multiple pieces of similar data, the database creates only one execution plan. This improves compilation and optimization efficiency.

The prerequisite is that the **customer_t1** table has been created by executing the preceding basic SQL statements. Execute the prepared statement to insert data in batches as follows:

- Step 1** Create the prepared statement object **pst** by calling the prepareStatement method of the Connection API.

```
PreparedStatement pst = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?,?)");
```

Step 2 Call the corresponding API to set parameters for each piece of data and call `addBatch` to add SQL statements to the batch processing.

```
for (int i = 0; i < 3; i++) {  
    pst.setInt(1, i);  
    pst.setString(2, "data " + i);  
    pst.addBatch();  
}
```

Step 3 Perform batch processing by calling the `executeBatch` method of the `PreparedStatement` API.

```
pst.executeBatch();
```

Step 4 Close the prepared statement object `pst` by calling the `close` method of the `PreparedStatement` API.

```
pst.close();
```

----End

NOTE

Do not terminate a batch processing action when it is ongoing; otherwise, database performance will deteriorate. Therefore, disable automatic commit during batch processing. Manually commit several lines at a time. The statement for disabling automatic commit is as follows:

```
conn.setAutoCommit(false);
```

Executing a Prepared Statement for Update

Prepared statements are compiled and optimized once but can be used in different scenarios by assigning different values. Using prepared statements improves execution efficiency. If you want to run a statement for several times, use a prepared statement.

The prerequisite is that the preceding prepared statement has been executed and data has been inserted into the `customer_t1` table in batches. Update data by running a prepared SQL statement as follows:

Step 1 Create the prepared statement object `pstmt` by calling the `prepareStatement` method of the `Connection` API.

```
PreparedStatement pstmt = conn.prepareStatement("UPDATE customer_t1 SET c_customer_name = ?  
WHERE c_customer_sk = 1");
```

Step 2 Set parameters by calling the `setString` method of the `PreparedStatement` API.

```
pstmt.setString(1, "new Data");
```

Step 3 Execute the prepared statement by calling the `executeUpdate` method of the `PreparedStatement` API.

```
int rowcount = pstmt.executeUpdate();
```

Step 4 Close the prepared statement object `pstmt` by calling the `close` method of the `PreparedStatement` API.

```
pstmt.close();
```

----End

NOTICE

After binding parameters are set in `prepareStatement`, a B packet or U packet is constructed and sent to the server when the SQL statement is executed. However, the maximum length of a B packet or U packet cannot exceed 1023 MB. If the data bound at a time is too large, an exception may occur because the packet is too long. Therefore, when setting binding parameters in `prepareStatement`, you need to evaluate and control the size of the bound data to avoid exceeding the upper packet limit.

Creating and Calling a Stored Procedure

GaussDB can call stored procedures through JDBC. The prerequisite is that the database connection is established using the connection object `conn`.

Create the **testproc** stored procedure as follows:

```
// Create the stored procedure (containing the out parameter) in the database as follows:
CREATE or REPLACE procedure testproc
(
    psv_in1 in integer,
    psv_in2 in integer,
    psv_inout inout integer
)
AS
BEGIN
    psv_inout := psv_in1 + psv_in2 + psv_inout;
END;
/
```

Call the **testproc** stored procedure as follows:

Step 1 Create a call statement object **cstmt** by calling the `prepareCall` method of `Connection`.

```
CallableStatement cstmt = conn.prepareCall("{? = CALL testproc(?,?,?)}");
```

Step 2 Set parameters by calling the `setInt` method of `CallableStatement`.

```
cstmt.setInt(2, 50);
cstmt.setInt(1, 20);
cstmt.setInt(3, 90);
```

Step 3 Register an output parameter by calling the **registerOutParameter** method in **CallableStatement**.

```
cstmt.registerOutParameter(4, Types.INTEGER); // Register an OUT parameter of the integer type.
```

Step 4 Execute SQL statements by calling the `execute` method of `CallableStatement`.

```
cstmt.execute();
```

Step 5 Obtain the out parameter by calling the `getInt` method of `CallableStatement`.

```
int out = cstmt.getInt(4);
```

Step 6 Close the call statement object **cstmt** by calling the `close` method of `CallableStatement`.

```
cstmt.close();
```

----End

 NOTE

- Some JDBC drivers support named parameters, which can be used to set parameters by name rather than sequence. If a parameter has the default value, you do not need to specify any parameter value but can use the default value directly. Even though the parameter sequence changes during a stored procedure, the application does not need to be modified. Currently, the GaussDB JDBC driver does not support this method.
- GaussDB does not support functions containing OUT parameters, or stored procedures and function parameters containing default values.
- When `conn.prepareStatement("{? = CALL testproc(?,?,?)}")` is used to execute a stored procedure, you can bind parameters in the sequence of placeholders. The first parameter is registered as an output parameter. You can also bind parameters based on the parameter sequence in the stored procedure and register the fourth parameter as an output parameter. In this example, the fourth parameter is registered as an output parameter.
- If JDBC is used to call a stored procedure whose returned value is a cursor, the returned cursor cannot be used.
- A stored procedure and a basic SQL statement must be run separately.
- Output parameters must be registered for parameters of the inout type in the stored procedure.

Adding Single-Shard Execution Syntaxes to Statements

The prerequisite is that the database has been connected using the `conn` connection object, the `test` table has been created, and data has been inserted into the table.

- Step 1** Call the `setClientInfo(String name,String value)` method of the Connection object to set the **nodeName** parameter.

```
conn.setClientInfo("nodeName","datanode1");
```

- Step 2** Execute the SQL statements by using the `executeQuery(String sql)` and `execute(String sql)` methods in Statement and the `executeQuery()` and `execute()` methods in PreparedStatement.

```
PreparedStatement pstmt = conn.prepareStatement("SELECT * FROM test");  
pstmt.execute();  
pstmt.executeQuery();  
Statement stmt=conn.createStatement();  
stmt.execute("SELECT * FROM test");  
stmt.executeQuery("SELECT * FROM test");
```

- Step 3** Set the parameter to an empty string to disable it.

```
conn.setClientInfo("nodeName","");
```

NOTICE

- This function is adapted based on the single-shard execution function of the kernel. Therefore, before using this function, check whether the database kernel supports single-shard execution.
- If the **nodeName** parameter is enabled, you need to manually disable the parameter after executing SQL statements. Otherwise, the execution of other query statements will be affected.
- Once the parameter is enabled, all statements of the current connection will be executed on the specified DN.
- After the parameter is enabled, the cache mechanism of PreparedStatement will be affected, cached statements will be cleared, and subsequent statements executed for single-shard queries will not be cached until the parameter is disabled.
- The parameter is a connection parameter. Therefore, the parameter value takes effect once. The API cannot be used to execute the statements on different shards at the same time.

----End

5.4.2.4 Processing Data in a Result Set

After running SQL statements, you need to process the result set. This section describes how to set the result set type, locate the result set, obtain the cursor position from a result set, and obtain data from the result set.

Setting a Result Set Type

Different types of result sets apply to different application scenarios. Applications select proper types of result sets based on the actual situation. Before running an SQL statement, you must create a statement object. Some methods of creating statement objects can set the type of a result set. The `java.sql.Connection` API provides the following three methods for creating statement objects:

```
// Create a Statement object. This object will generate a ResultSet object with a specified type and
concurrency.
createStatement(int resultSetType, int resultSetConcurrency);

// Create a PreparedStatement object. This object will generate a ResultSet object with a specified type and
concurrency.
prepareStatement(String sql, int resultSetType, int resultSetConcurrency);

// Create a CallableStatement object. This object will generate a ResultSet object with a specified type and
concurrency.
prepareCall(String sql, int resultSetType, int resultSetConcurrency);
```

Table 5-10 Result set types

Parameter	Description
resultSetType	<p>Type of a result set. There are three types of result sets:</p> <ul style="list-style-type: none"> • ResultSet.TYPE_FORWARD_ONLY: The ResultSet object can only be navigated forward. It is the default value. • ResultSet.TYPE_SCROLL_SENSITIVE: You can view the modified result by scrolling to the modified row. • ResultSet.TYPE_SCROLL_INSENSITIVE: The ResultSet object is insensitive to changes in the underlying data source. <p>NOTE After a result set has obtained data from the database, the result set is insensitive to data changes made by other transactions, even if the result set type is ResultSet.TYPE_SCROLL_SENSITIVE. To obtain up-to-date data of the record pointed by the cursor from the database, call the refreshRow() method in a ResultSet object.</p>
resultSetConcurrency	<p>Concurrency type of a result set. There are two types of concurrency.</p> <ul style="list-style-type: none"> • ResultSet.CONCUR_READ_ONLY: Data in a result set cannot be updated except that an updated statement has been created in the result set data. • ResultSet.CONCUR_UPDATEABLE: changeable result set. The concurrency type for a result set object can be updated if the result set is scrollable.

Positioning a Cursor in a Result Set

ResultSet objects include a cursor pointing to the current data row. The cursor is initially positioned before the first row. The next method moves the cursor to the next row from its current position. When a ResultSet object does not have a next row, a call to this method returns **false**. Therefore, this method is used in the while loop for result set iteration. However, the JDBC driver provides more cursor positioning methods for scrollable result sets, which allows positioning cursor in the specified row. [Table 5-11](#) describes these methods.

Table 5-11 Methods for positioning in a result set

Method	Description
next()	Moves cursor to the next row from its current position.
previous()	Moves cursor to the previous row from its current position.
beforeFirst()	Places cursor before the first row.
afterLast()	Places cursor after the last row.

Method	Description
first()	Places cursor to the first row.
last()	Places cursor to the last row.
absolute(int row)	Places cursor to a specified row.
relative(int rows)	Moves the result set downwards by the number of rows specified by rows when rows is set to a positive number, or moves the result set upwards by the number of rows specified by rows when rows is set to a negative number.

Obtaining the Cursor Position from a Result Set

This cursor positioning method can be used to change the cursor position for a scrollable result set. The JDBC driver provides a method to obtain the cursor position in a result set. [Table 5-12](#) describes these methods.

Table 5-12 Methods for obtaining a cursor position in a result set

Method	Description
isFirst()	Checks whether it is in the first row.
isLast()	Checks whether it is in the last row.
isBeforeFirst()	Checks whether it is before the first row.
isAfterLast()	Checks whether it is after the last row.
getRow()	Obtains its current row number.

Obtaining Data from a Result Set

ResultSet objects provide a variety of methods to obtain data from a result set. [Table 5-13](#) describes the common methods for obtaining data. If you want to know more about other methods, see JDK official documents.

Table 5-13 Common methods for obtaining data from a result set

Method	Description
getInt(int columnIndex)	Retrieves the value of the column designated by a column index in the current row as an integer.

Method	Description
<code>getInt(String columnLabel)</code>	Retrieves the value of the column designated by a column label in the current row as an integer.
<code>getString(int columnIndex)</code>	Retrieves the value of the column designated by a column index in the current row as a string.
<code>getString(String columnLabel)</code>	Retrieves the value of the column designated by a column label in the current row as a string.
<code>getDate(int columnIndex)</code>	Retrieves the value of the column designated by a column index in the current row as a date.
<code>getDate(String columnLabel)</code>	Retrieves the value of the column designated by a column name in the current row as a date.

5.4.2.5 Closing a Connection

After you complete required data operations in the database, close the database connection.

You can call the close method to close the connection.

```
conn.close();
```

NOTE

Many database classes such as Connection, Statement, and ResultSet have a close() method. Close these classes after using their objects. Closing Connection objects is to close all the related Statement objects, and closing a Statement object is to close its ResultSet object.

5.4.3 Typical Application Development Examples

The Java code examples in this section use the **opengaussjdbc.jar** package by default.

5.4.3.1 Configuring Connection Parameters in Different Scenarios

NOTE

In the following example, **host:port** represents a node, where **host** indicates the name or IP address of the server where the database resides, and **port** indicates the port number of the server where the database resides.

DR

A customer has two database clusters. Cluster A is the production cluster, and cluster B is the DR cluster. When the customer performs a DR failover, cluster A

becomes the DR cluster, and cluster B becomes the production cluster. In this case, to avoid application restart or re-release caused by configuration file modification, the customer can write clusters A and B to the connection string when initializing the configuration file. In this case, if the primary cluster cannot be connected, the driver attempts to connect to the DR cluster. Assume that cluster A is {node1,node2,node3} and cluster B is {node4,node5,node6}.

Take the dual-cluster Dorado as an example. To connect to the new primary cluster after the switchover, you can set the URL as follows:

```
jdbc:opengauss://node1,node2,node3,node4,node5,node6/database
```

Load Balancing

A customer has a database cluster that contains the following nodes: {node1,node2,node3,node4,node5,node6,node7,node8,node9,node10,node11,node12}.

- The customer establishes 120 persistent connections in application A and expects that the connections on application A can be evenly distributed on each node in the current cluster. The URL can be configured as follows:

```
jdbc:opengauss://node1,node2,node3/database?autoBalance=true
```
- The customer develops two applications B and C and expects that the three applications to be evenly distributed on specified nodes. For example, the connections of application A are distributed on {node1,node2,node3,node4}; the connections of application B are distributed on {node5,node6,node7,node8}; and the connections of application C are distributed on {node9,node10,node11,node12}. The URLs can be configured as follows (If the first four nodes in the URL cannot be connected, the application connects to the fifth node based on the information in the pgxc_node system catalog. All available nodes in the cluster are connected in polling mode.):

Application A:

```
jdbc:opengauss://node1,node2,node3,node4,node5/database?autoBalance=priority4
```

Application B:

```
jdbc:opengauss://node5,node6,node7,node8,node9/database?autoBalance=priority4
```

Application C:

```
jdbc:opengauss://node9,node10,node11,node12,node1/database?autoBalance=priority4
```

- The customer develops more applications, uses the same connection configuration string, and expects that the application connections can be evenly distributed on each node in the cluster. The URL can be configured as follows:

```
jdbc:opengauss://node1,node2,node3,node4/database?autoBalance=shuffle
```

- If the customer does not want to use the load balancing function, configure the URL as follows:

```
jdbc:opengauss://node1/database
```

Or

```
jdbc:opengauss://node1/database?autoBalance=false
```

NOTE

When the **autoBalance** parameter is enabled, the interval for the JDBC to refresh the available CN list is 10s by default. You can use **refreshCNIPListTime** to set the interval:

```
jdbc:opengauss://node1,node2,node3,node4/database?autoBalance=true&refreshCNIPListTime=3
```

Log Diagnosis Scenario

If you encounter slow data import or some errors that are difficult to analyze, the trace log function can be enabled for diagnosis. The URL can be configured as follows:

```
jdbc:opengauss://node1/database?loggerLevel=trace
```

High Performance

A customer may execute the same SQL statement for multiple times with different input parameters. To improve the execution efficiency, the **prepareThreshold** parameter can be enabled to avoid repeatedly generating execution plans. The URL can be configured as follows:

```
jdbc:opengauss://node1/database?prepareThreshold=5
```

A customer queries 10 million data records at a time. To prevent memory overflow caused by simultaneous return of the data records, the **defaultRowFetchSize** parameter can be used. The URL can be configured as follows:

```
jdbc:opengauss://node1/database?defaultRowFetchSize=50000
```

A customer needs to insert 10 million data records in batches. To improve efficiency, the **batchMode** parameter can be used. The URL can be configured as follows:

```
jdbc:opengauss://node1/database?batchMode=on
```

5.4.3.2 Creating and Calling a Stored Procedure

It also demonstrates how to connect to a database and create and call stored procedures.

```
// The following uses opengaussjdbc.jar as an example.  
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.  
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).  
// You need to change the values of $ip, $port, and database.
```

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.sql.CallableStatement;  
import java.sql.Types;  
  
public class DBTest {  
  
    // Create a database connection.  
    public static Connection GetConnection(String username, String passwd) {  
        String driver = "com.huawei.opengauss.jdbc.Driver";  
        String sourceURL = "jdbc:opengauss://$ip:$port/database";  
        Connection conn = null;  
        try {  
            // Load the database driver.  
            Class.forName(driver);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
        return null;
    }

    try {
        // Create a database connection.
        conn = DriverManager.getConnection(sourceURL, username, passwd);
        System.out.println("Connection succeed!");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }

    return conn;
};

// Create a stored procedure.
public static void CreateCallable(Connection conn) {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();
        // Create a stored procedure and return the sum of the three input values.
        stmt.execute("create or replace procedure testproc \n" +
            "\n" +
            "    psv_in1 in integer,\n" +
            "    psv_in2 in integer,\n" +
            "    psv_inout inout integer\n" +
            ")\n" +
            "as\n" +
            "begin\n" +
            "    psv_inout := psv_in1 + psv_in2 + psv_inout;\n" +
            "end;\n" +
            "/");
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } finally {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

// Call the stored procedure.
public static void ExecCallableSQL(Connection conn) {
    CallableStatement cstmt = null;
    try {
        cstmt=conn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
        cstmt.setInt(2, 50);
        cstmt.setInt(1, 20);
        cstmt.setInt(3, 90);
        cstmt.registerOutParameter(4, Types.INTEGER); // Register an OUT parameter of the integer type.
        cstmt.execute();
        int out = cstmt.getInt(4); // Obtain the OUT parameter.
        System.out.println("The CallableStatment TESTPROC returns:"+out);
        cstmt.close();
    } catch (SQLException e) {
        if (cstmt != null) {
            try {
                cstmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}
```

```
}

/**
 * Main process. Call static methods one by one.
 * @param args
 */
public static void main(String[] args) {
    // Create a database connection.
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");
    Connection conn = GetConnection(userName, password);

    // Create a stored procedure.
    CreateCallable(conn);

    // Call the stored procedure.
    ExecCallableSQL(conn);

    // Close the database connection.
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

The execution result of the preceding example is as follows:

```
Connection succeed!
The CallableStatement TESTPROC returns:160
```

5.4.3.3 Obtaining the Return Value of a Function

JDBC obtains the return value when calling a function. The following example shows the return values of the bit and float8 types. For other data types, refer to this example.

```
// The following uses opengaussjdbc.jar as an example.
// There will be security risks if the username and password used for authentication are directly written into
// code. It is recommended that the username and password be stored in the configuration file or
// environment variables (the password must be stored in ciphertext and decrypted when being used) to
// ensure security.
// In this example, the username and password are stored in environment variables. Before running this
// example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local
// environment (set the environment variable names based on the actual situation).
// You need to change the values of $ip, $port, and database.

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.CallableStatement;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.Types;

public class Type {
    public static void main(String[] args) throws SQLException {
        String driver = "com.huawei.opengauss.jdbc.Driver";
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String sourceURL = "jdbc:opengauss://$ip.$port|database";
        Connection conn = null;

        try {
```

```
// Load the database driver.
Class.forName(driver);
} catch (Exception e) {
    e.printStackTrace();
}

try {
    // Establish a database connection in non-encrypted mode.
    conn = DriverManager.getConnection(sourceURL, username, passwd);
    System.out.println("Connection succeed!");
} catch (Exception e) {
    e.printStackTrace();
}

// Create a table.
String createsql = "create table if not exists t_bit(col_bit bit, col_bit1 int)";
Statement stmt = conn.createStatement();
stmt.execute(createsql);
stmt.close();
// Example of using the bit type. Note that the value range of the bit type is [0,1].
Statement st = conn.createStatement();
String sqlstr = "create or replace function fun_1()\n" + "returns bit AS $$\n"
    + "select col_bit from t_bit limit 1;\n" + "$$\n" + "LANGUAGE SQL;";
st.execute(sqlstr);
CallableStatement c = conn.prepareCall("{ ? = call fun_1() }");
// Register the output type, which is a bit string.
c.registerOutParameter(1, Types.BIT);
c.execute();
// Use the Boolean type to obtain the result.
System.out.println(c.getBoolean(1));

// Example of using the float8 type.
st.execute("create table if not exists t_float(col1 float8,col2 int)");
PreparedStatement pstmt = conn.prepareStatement("insert into t_float values(?)");
pstmt.setDouble(1, 123456.123);
pstmt.execute();
pstmt.close();

// Example of using the function whose return value is of the float8 type.
st.execute(
    "create or replace function func_float() " + "return float8 " + "as declare " + "var1 float8; " + "begin
"
    + " select col1 into var1 from t_float limit 1; " + " return var1; " + "end;");
CallableStatement cs = conn.prepareCall("{? = call func_float()}");
cs.registerOutParameter(1, Types.DOUBLE);
cs.execute();
System.out.println(cs.getDouble(1));
st.close();

// Close the connection to the database.
try {
    conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

The execution result of the preceding example is as follows:

```
Connection succeed!
false
123456.123
```

5.4.3.4 Batch Query

In this example, `setFetchSize` adjusts the memory usage of the client by using the database cursor to obtain server data in batches. It may increase network

interaction and affect performance to some extent. The cursor is valid within a transaction. Therefore, disable automatic transaction commit and then manually commit the transaction.

```
// The following uses opengaussjdbc.jar as an example.
// There will be security risks if the username and password used for authentication are directly written into
// code. It is recommended that the username and password be stored in the configuration file or
// environment variables (the password must be stored in ciphertext and decrypted when being used) to
// ensure security.
// In this example, the username and password are stored in environment variables. Before running this
// example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local
// environment (set the environment variable names based on the actual situation).
// You need to change the values of $ip, $port, and database.

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.PreparedStatement;

public class Batch {
    public static void main(String[] args) throws SQLException {
        String driver = "com.huawei.opengauss.jdbc.Driver";
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String sourceURL = "jdbc:opengauss://$ip:$port/database";
        Connection conn = null;

        try {
            // Load the database driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            // Establish a database connection in non-encrypted mode.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Disable automatic commit.
        conn.setAutoCommit(false);

        // Create a table.
        Statement st = conn.createStatement();
        st.execute("create table mytable (col1 int);");

        // Insert 200 rows of data into the table.
        PreparedStatement pstmt = conn.prepareStatement("insert into mytable values (?)");
        for (int i = 0; i < 200; i++) {
            pstmt.setInt(1, i + 1);
            pstmt.addBatch();
        }
        pstmt.executeBatch();
        conn.commit();
        pstmt.close();

        // Open the cursor and obtain 50 rows of data each time.
        st.setFetchSize(50);
        int fetchCount = 0;
        ResultSet rs = st.executeQuery("SELECT * FROM mytable");
        while (rs.next()) {
            fetchCount++;
        }
        System.out.println(fetchCount == 200);
    }
}
```

```
conn.commit();
rs.close();

// Disable the server cursor.
st.setFetchSize(0);
fetchCount = 0;
rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next()) {
    fetchCount++;
}
System.out.println(fetchCount == 200);
conn.commit();
rs.close();

// Close the statement object and database connection.
st.close();
conn.close();

}
}
```

The execution result of the preceding example is as follows:

```
Connection succeed!
true
true
```

Enable automatic commit.

```
conn.setAutoCommit(true);
```

5.4.3.5 SQL Retry at an Application Layer

If the primary DN is faulty and cannot be restored within 10s, the standby DN is automatically promoted to primary to ensure the normal running of the GaussDB cluster. Jobs running during the failover will fail and those started after the failover will not be affected. To prevent upper-layer services from being affected by the DN failover, refer to the following example to construct an SQL retry mechanism at the service layer.

```
// The following uses opengaussjdbc.jar as an example.
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
// You need to change the values of $ip, $port, and database.
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

class ExitHandler extends Thread {
    private Statement cancel_stmt = null;

    public ExitHandler(Statement stmt) {
        super("Exit Handler");
        this.cancel_stmt = stmt;
    }

    public void run() {
        System.out.println("exit handle");
        try {
```

```
        this.cancel_stmt.cancel();
    } catch (SQLException e) {
        System.out.println("cancel query failed.");
        e.printStackTrace();
    }
}
}

public class SQLRetry {
    // Establish a database connection in non-encrypted mode.
    public static Connection GetConnection(String username, String passwd) {
        String driver = "com.huawei.opengauss.jdbc.Driver";
        String sourceURL = "jdbc:opengauss://$ip:$port/database";
        Connection conn = null;
        try {
            // Load the database driver.
            Class.forName(driver).newInstance();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        try {
            // Create a database connection.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        return conn;
    }

    // Execute common SQL statements to create the jdbc_test1 table.
    public static void CreateTable(Connection conn) {
        Statement stmt = null;
        try {
            stmt = conn.createStatement();

            Runtime.getRuntime().addShutdownHook(new ExitHandler(stmt));

            // Execute common SQL statements.
            int rc2 = stmt
                .executeUpdate("DROP TABLE if exists jdbc_test1;");

            int rc1 = stmt
                .executeUpdate("CREATE TABLE jdbc_test1(col1 INTEGER, col2 VARCHAR(10));");

            stmt.close();
        } catch (SQLException e) {
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException e1) {
                    e1.printStackTrace();
                }
            }
            e.printStackTrace();
        }
    }

    // Execute a prepared statement to insert data in batches.
    public static void BatchInsertData(Connection conn) {
        PreparedStatement pst = null;

        try {
            // Generate a prepared statement.

```

```
pst = conn.prepareStatement("INSERT INTO jdbc_test1 VALUES (?,?)");
for (int i = 0; i < 100; i++) {
    // Add parameters.
    pst.setInt(1, i);
    pst.setString(2, "data " + i);
    pst.addBatch();
}
// Perform batch processing.
pst.executeBatch();
pst.close();
} catch (SQLException e) {
    if (pst != null) {
        try {
            pst.close();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    }
    e.printStackTrace();
}
}

// Run a prepared statement to update data.
private static boolean QueryRedo(Connection conn){
    PreparedStatement pstmt = null;
    boolean retValue = false;
    try {
        pstmt = conn
            .prepareStatement("SELECT col1 FROM jdbc_test1 WHERE col2 = ?");

        pstmt.setString(1, "data 10");
        ResultSet rs = pstmt.executeQuery();

        while (rs.next()) {
            System.out.println("col1 = " + rs.getString("col1"));
        }
        rs.close();

        pstmt.close();
        retValue = true;
    } catch (SQLException e) {
        System.out.println("catch..... retValue " + retValue);
        if (pstmt != null) {
            try {
                pstmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

System.out.println("finesh.....");
return retValue;
}

// Configure the number of retry attempts for the retry of a query statement upon a failure.
public static void ExecPreparedSQL(Connection conn) throws InterruptedException {
    int maxRetryTime = 10;
    int time = 0;
    String result = null;
    do {
        time++;
        try {
            System.out.println("time:" + time);
            boolean ret = QueryRedo(conn);
            if (ret == false){
                System.out.println("retry, time:" + time);
                Thread.sleep(10000);
            }
        }
    }
}
```

```
QueryRedo(conn);
}
    } catch (Exception e) {
        e.printStackTrace();
    }
} while (null == result && time < maxRetryTime);

}

/**
 * Main program. Call static methods one by one.
 * @param args
 * @throws InterruptedException
 */
public static void main(String[] args) throws InterruptedException {
    // Create a database connection.
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");
    Connection conn = GetConnection(userName, password);

    // Create a table.
    CreateTable(conn);

    // Insert data in batches.
    BatchInsertData(conn);

    // Run a prepared statement to update data.
    ExecPreparedSQL(conn);

    // Close the database connection.
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

The execution result of the preceding example is as follows:

```
Connection succeed!
time:1
col1 = 10
finesh.....
time:2
col1 = 10
finesh.....
time:3
col1 = 10
finesh.....
time:4
col1 = 10
finesh.....
time:5
col1 = 10
finesh.....
time:6
col1 = 10
finesh.....
time:7
col1 = 10
finesh.....
time:8
col1 = 10
finesh.....
time:9
col1 = 10
finesh.....
```

```
time:10  
col1 = 10  
finesh.....  
exit handle
```

5.4.3.6 Logical Replication

The example demonstrates how to use the logical replication function through the JDBC APIs.

Configuration Options

For details about the configuration options of logical replication, see section "Logical Replication > Logical Decoding" in *Feature Guide*. In addition, the following configuration items are added for streaming decoding tools such as JDBC:

1. Decoding thread concurrency

Set **parallel-decode-num** to specify the number of decoder threads for parallel decoding. The value is an integer ranging from 1 to 20. The value **1** indicates that decoding is performed based on the original serial logic. Other values indicate that parallel decoding is enabled. The default value is **1**. When this parameter is set to **1**, do not configure the following options: **decode-style**, **sending-batch**, and **parallel-queue-size**.

2. Decoding format

Configure **decode-style** to specify the decoding format. The value can be 'j', 't' or 'b' of the char type, indicating the JSON, text, or binary format, respectively. The default value is 'b', indicating binary decoding. This option is set only when parallel decoding is allowed and binary decoding is supported only in the parallel decoding scenario. For the JSON and TEXT formats, in the decoding result sent in batches, the uint32 consisting of the first four bytes of each decoding statement indicates the total number of bytes of the statement (the four bytes occupied by the uint32 are excluded, and **0** indicates that the decoding of this batch ends). The 8-byte uint64 indicates the corresponding LSN (**begin** corresponds to **first_lsn**, **commit** corresponds to **end_lsn**, and other values correspond to the LSN of the statement).

 NOTE

The binary encoding rules are as follows:

1. The first four bytes represent the total number of bytes of the decoding result of statements following the statement-level delimiter letter P (excluded) or the batch end character F (excluded). If the value is 0, the decoding of this batch ends.
2. The next eight bytes (uint64) indicate the corresponding LSN (**begin** corresponds to **first_lsn**, **commit** corresponds to **end_lsn**, and other values correspond to the LSN of the statement).
3. The next one-byte letter can be **B**, **C**, **I**, **U**, or **D**, representing BEGIN, COMMIT, INSERT, UPDATE, or DELETE.
4. If **B** is used in the step 3:
 1. The next eight bytes (uint64) indicate the CSN.
 2. The next eight bytes (uint64) indicate **first_lsn**.
 3. (Optional) If the next 1-byte letter is **T**, the following four bytes (uint32) indicate the timestamp length for committing the transaction. The following characters with the same length are the timestamp character string.
 4. (Optional) If the next one-byte letter is **N**, the following four bytes (uint32) indicate the length of the transaction username. The following characters with the same length are the transaction username.
 5. There may still be a decoding statement subsequently, with a 1-byte letter **P** or **F** as a separator between statements. **P** indicates that there are still decoding statements in this batch, and **F** indicates that decoding in this batch is complete.
5. If **C** is used in the step 3:
 1. (Optional) If the next 1-byte letter is **X**, the following eight bytes (uint64) indicate XID.
 2. (Optional) If the next 1-byte letter is **T**, the following four bytes (uint32) indicate the timestamp length. The following characters with the same length are the timestamp character string.
 3. When logs are sent in batches, decoding results of other transactions may still exist after a COMMIT log is decoded. If the next 1-byte letter is **P**, the batch still needs to be decoded. If the letter is **F**, the batch decoding ends.
6. If **I**, **U**, or **D** is used in the step 3:
 1. The following two bytes (uint16) indicate the length of the schema name.
 2. The schema name is read based on the preceding length.
 3. The following two bytes (uint16) indicate the length of the table name.
 4. The table name is read based on the preceding length.
 5. (Optional) If the next 1-byte letter is **N**, it indicates a new tuple. If the letter is **O**, it indicates an old tuple. In this case, the new tuple is sent first.
 1. The following two bytes (uint16) indicate the number of columns to be decoded for the tuple, which is recorded as **attrnum**.
 2. The following procedure is repeated for *attrnum* times:
 1. The next two bytes (uint16) indicate the length of the column name.
 2. The column name is read based on the preceding length.
 3. The following four bytes (uint32) indicate the OID of the current column type.
 4. The next four bytes (uint32) indicate the length of the value (stored in the character string format) in the current column. If the value is **0xFFFFFFFF**, it indicates null. If the value is **0**, it indicates a character string whose length is 0.
 5. The column value is read based on the preceding length.

6. Because there may still be a decoding statement after, if the next one-byte letter is **P**, it indicates that the batch still needs to be decoded, and if the next one-byte letter is **F**, it indicates that decoding of the batch ends.
3. Decoding only on the standby node
Configure the **standby-connection** option to specify whether to perform decoding only on the standby node. The value is of the Boolean type (**0** or **1**). The value **true** (or **1**) indicates that only the standby node can be connected for decoding. When the primary node is connected for decoding, an error is reported and the system exits. The value **false** (or **0**) indicates that there is no restriction. The default value is **false** (or **0**).
4. Batch sending
Configure **sending-batch** to specify whether to send results in batches. The value is an integer ranging from 0 to 1. The value **0** indicates that decoding results are sent one by one. The value **1** indicates that decoding results are sent in batches when the accumulated size of decoding results reaches 1 MB. The default value is **0**. This parameter can be set only during parallel decoding. In the scenario where batch sending is enabled, if the decoding format is 'j' or 't', before each original decoding statement, a uint32 type is added indicating the length of the decoding result (excluding the current uint32 type), and a uint64 type is added, indicating the LSN corresponding to the current decoding result.
5. Length of the parallel decoding queue
Configure **parallel-queue-size** to specify the length of the queue for interaction among parallel logical decoding threads. The value ranges from 2 to 1024 and must be a power of 2. The default value is **128**. The queue length is positively correlated with the memory usage during decoding.
6. Memory threshold for logical decoding
The **max-txn-in-memory** configuration item specifies the memory threshold for caching the intermediate decoding result of a single transaction, in MB. In parallel decoding mode, this parameter does not take effect. In serial decoding mode, the value range is [0,100]. The default value is **0**, indicating that the memory usage is not controlled.
The **max-reorderbuffer-in-memory** configuration item specifies the memory threshold for caching intermediate decoding results of all transactions, in GB. In serial decoding mode, the value range is [0,100]. The default value is **0**, indicating that the memory usage is not controlled. In parallel decoding mode, the value ranges from 1 to 50% of the value of **max_process_memory**. The default value is the larger value between 1 and **max_process_memory**/1048576 x 10%. **1048576** indicates the unit conversion from KB to GB. When the memory usage exceeds the threshold, intermediate decoding results are written into a temporary file during decoding, affecting the logical decoding performance. The size of the temporary file is proportional to the amount of data modified by the transaction. If there are too many temporary files, the disk may enter the read-only state.
7. Logical decoding sending timeout threshold
The **sender-timeout** configuration item specifies the heartbeat timeout threshold between the kernel and client. If no message is received from the client within the period, the logical decoding stops and disconnects from the client. The unit is ms, and the value range is [0,2147483647]. The default value depends on the value of the GUC parameter **logical_sender_timeout**.

The value **0** indicates that logical decoding does not proactively disconnect from the client. A small value, for example, 1 ms, indicates that decoding tasks may be interrupted.

8. User blacklist options for logical decoding

Use the user blacklist for logical decoding. The transaction operations of blacklisted users are filtered from the logical decoding result. The options are as follows:

- a. **exclude-userids**: specifies the OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not check whether the user OIDs exist. The OIDs of the same service user on different DNs may be different. Therefore, the OID of the service user on each DN needs to be transferred for logical decoding of directly connected DNs in distributed mode. Otherwise, the logical decoding results of some DNs may be filtered while those of some DNs are not filtered.
- b. **exclude-users**: specifies blacklisted usernames. Multiple usernames are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify usernames. If the decoding is interrupted because the user does not exist and the corresponding blacklisted user does not exist at the time when logs are generated, you can set **dynamic-resolution** to **true** or delete the username from the blacklist to start decoding and continue to obtain logical logs.
- c. **dynamic-resolution**: specifies whether to dynamically parse blacklisted usernames. The default value is **true**. If the parameter is set to **false**, an error is reported and the logical decoding exits when the decoding detects that the user does not exist in blacklist **exclude-users**. If the parameter is set to **true**, decoding continues when it detects that the user does not exist in blacklist **exclude-users**.

9. Output options for transaction logic logs

- a. **include-xids**: specifies whether the BEGIN logical log of a transaction outputs the transaction ID. The default value is **true**.
- b. **include-timestamp**: specifies whether the BEGIN logical log of a transaction outputs the time when the transaction is committed. The default value is **false**.
- c. **include-user**: specifies whether the BEGIN logical log of a transaction outputs the username of the transaction. The default value is **false**. The username of a transaction refers to the authorized user, that is, the login user who executes the session corresponding to the transaction. The username does not change during the execution of the transaction.

10. By default, **socketTimeout** for the logical decoding connection in JDBC is set to **10s**. When the primary node is overloaded during decoding on the standby node, the connection may be closed due to timeout. You can set **withStatusInterval(10000,TimeUnit.MILLISECONDS)** to adjust the timeout interval and retain the connection.

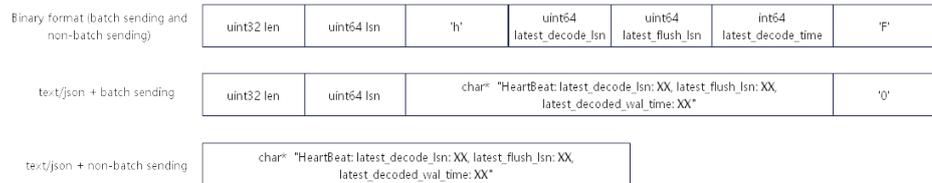
11. Heartbeat log output option

enable-heartbeat: specifies whether to generate heartbeat logs. The default value is **false**.

 NOTE

The following information is about parsing heartbeat logs with figure showing the format of the logs (in consideration of forward compatibility, the LSN naming mode is retained. The actual meaning depends on the actual scenario):

- In the binary format, the message starts with the character 'h', indicating a heartbeat log.
- The following is the heartbeat log content. For the first 8-byte uint64 string, in the decoding scenario where DNs are directly connected, this string is an LSN, indicating the end position of the WAL read when the heartbeat logical log is sent; in the decoding scenario where distributed strong consistency is required, this string is a CSN, indicating the decoding log transaction CSN that has been sent when the heartbeat logical log is sent. For the second 8-byte uint64 string, in the decoding scenario where DNs are directly connected, this string is an LSN, indicating the location of the WAL that has been flushed to disks when the heartbeat logical log is sent; in the decoding scenario where distributed strong consistency is required, this string is a CSN, indicating the CSN to be obtained by the next transaction committed by the cluster. The last 8-byte int64 string indicates the generation timestamp (starting from January 1, 1970) of the latest decoded transaction log or checkpoint log.
- Message terminator: If in binary format, it is 'F'. If the format is TEXT or JSON and the message is sent in batches, the terminator is '0'. Otherwise, there is no terminator. The message content is transmitted in big-endian mode.



The decoding performance (Xlog consumption) is greater than or equal to 100 Mbit/s in the following standard parallel decoding scenario: 16-core CPU, 128 GB memory, network bandwidth > 200 Mbit/s, 10 to 100 columns in a table, 0.1 KB to 1 KB data in a single row, INSERT as main DML operations, less than 4096 statements in a single transaction, **parallel-decode-num** set to **8**, decoding format as **'b'**, and batch sending function enabled. To ensure that the decoding performance meets the requirements and minimize the impact on services, you are advised to set up only one parallel decoding connection on a standby node to ensure that the CPU, memory, and bandwidth resources are sufficient.

Examples

Prerequisites for code running:

1. Add the IP address (for example, 10.11.12.34) of the JDBC user to the whitelist of the data replication permission. The command is as follows:


```
// Distributed strongly-consistent decoding on CNs (CN_IP indicates the IP address of the CN where a logical decoding task is started; to query it, run the cm_ctl query -Cvip command):
gs_guc reload -Z datanode -N all -I all -h 'host replication all CN_IP/24 trust'
gs_guc reload -Z coordinator -N all -I all -h 'host replication all 10.11.12.34/32 sha256'

//Distributed decoding on directly-connected DNs:
gs_guc reload -Z datanode -N all -I all -h 'host replication all 10.11.12.34/32 sha256'
```
2. Set **wal_level** to **logical**. For details, contact the administrator.
3. Create tables **t1** and **t2** and perform DDL or DML operations on the tables.


```
// The following uses opengaussjdbc.jar as an example.
// There will be security risks if the username and password used for authentication are directly written into
```

```
code. It is recommended that the username and password be stored in the configuration file or
environment variables (the password must be stored in ciphertext and decrypted when being used) to
ensure security.
// In this example, the username and password are stored in environment variables. Before running this
example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local
environment (variable names are subject to the actual situation).
// Change the values of $ip, $port, and database.
// The logical replication class PGReplicationStream is a non-thread-safe class. Concurrent calls may cause
data exceptions.

import com.huawei.opengauss.jdbc.PGProperty;
import com.huawei.opengauss.jdbc.jdbc.PgConnection;
import com.huawei.opengauss.jdbc.replication.LogSequenceNumber;
import com.huawei.opengauss.jdbc.replication.PGReplicationStream;

import java.nio.ByteBuffer;
import java.sql.DriverManager;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class LogicalReplicationDemo {
    private static PgConnection conn = null;
    public static void main(String[] args) {
        String driver = "com.huawei.opengauss.jdbc.Driver";
        // Configure the IP address and haPort number of the database. By default, the port number is the
port number of the connected CN or DN plus 1.
        String sourceURL = "jdbc:opengauss://$ip:$port/database";

        // The default name of the logical replication slot is replication_slot.
        // Test mode: Create a logical replication slot.
        int TEST_MODE_CREATE_SLOT = 1;
        // Test mode: Enable logical replication (the prerequisite is that the logical replication slot already
exists).
        int TEST_MODE_START_REPL = 2;
        // Test mode: Drop a logical replication slot.
        int TEST_MODE_DROP_SLOT = 3;
        // Enable different test modes.
        int testMode = TEST_MODE_START_REPL;

        try {
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        try {
            Properties properties = new Properties();
            PGProperty.USER.set(properties, System.getenv("EXAMPLE_USERNAME_ENV"));
            PGProperty.PASSWORD.set(properties, System.getenv("EXAMPLE_PASSWORD_ENV"));
            // For logical replication, the following three attributes are required:
            PGProperty.ASSUME_MIN_SERVER_VERSION.set(properties, "9.4");
            PGProperty.REPLICATION.set(properties, "database");
            PGProperty.PREFER_QUERY_MODE.set(properties, "simple");
            conn = (PgConnection) DriverManager.getConnection(sourceURL, properties);
            System.out.println("connection success!");

            if(testMode == TEST_MODE_CREATE_SLOT){
                conn.getReplicationAPI()
                    .createReplicationSlot()
                    .logical()
                    .withSlotName("replication_slot") // If the character string contains uppercase letters, the
uppercase letters are automatically converted to lowercase letters.
                    .withOutputPlugin("mppdb_decoding")
                    .make();
            }else if(testMode == TEST_MODE_START_REPL) {
                // Create a replication slot before enabling this mode.
                LogSequenceNumber waitLSN = LogSequenceNumber.valueOf("6F/E3C53568"); // LSN needs to
be modified based on the actual situation.
            }
        }
    }
}
```



```
BEGIN CSN: 2016 first_lsn: 0/2816D60
table public t1 DELETE: a[integer]:1 b[integer]:5 c[text]:'hello'
COMMIT XID: 15506
```

Example of the decoding result in JSON format (that is, 'j' format):

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
{"table_name":"public.t1","op_type":"INSERT","columns_name":["a","b","c"],"columns_type":
["integer","integer","text"],"columns_val":["1","2","hello"],"old_keys_name":[],"old_keys_type":
[],"old_keys_val":[]}
COMMIT XID: 15504
BEGIN CSN: 2015 first_lsn: 0/2816C20
{"table_name":"public.t1","op_type":"UPDATE","columns_name":["a","b","c"],"columns_type":
["integer","integer","text"],"columns_val":["1","5","hello"],"old_keys_name":["a","b","c"],"old_keys_type":
["integer","integer","text"],"old_keys_val":["1","2","hello"]}
COMMIT XID: 15505
BEGIN CSN: 2016 first_lsn: 0/2816D60
{"table_name":"public.t1","op_type":"DELETE","columns_name":[],"columns_type":[],"columns_val":
[],"old_keys_name":["a","b","c"],"old_keys_type":["integer","integer","text"],"old_keys_val":["1","5","hello"]}
COMMIT XID: 15506
```

5.4.4 JDBC Interface Reference

This section describes common JDBC APIs. For more APIs, check JDK 1.8 (software package) and JDBC 4.2.

5.4.4.1 java.sql.Connection

java.sql.Connection is an API for connecting to a database.

Table 5-14 Support status for java.sql.Connection

Method	Description	Return Type	throws	Support JDBC 4
abort(Executor executor)	Aborts an open connection.	void	SQLException	Yes
clearWarnings()	Clears all warnings reported for this connection object.	void	SQLException	Yes
close()	Releases the database and JDBC resources of this connection object immediately instead of waiting for them to be automatically released.	void	SQLException	Yes
commit()	Makes all changes made since the last commit/rollback permanent and releases any database locks currently held by this connection object. This method should be used only when the autocommit mode is disabled.	void	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
createArrayOf(String typeName, Object[] elements)	Creates an array object using this factory method.	Array	SQLException	Yes
createBlob()	Constructs an object that implements the BLOB API. The returned object does not contain data initially. The setBinaryStream and setBytes methods of the BLOB API can be used to add data to BLOBs.	Blob	SQLException	Yes
createClob()	Constructs an object that implements the CLOB API. The returned object does not contain data initially. The setAsciiStream, setCharacterStream, and setString methods of the CLOB API can be used to add data to CLOBs.	Clob	SQLException	Yes
createSQLXML()	Constructs an object that implements the SQLXML API. The returned object does not contain data initially. You can use the createXmlStreamWriter object of the SQLXML API and the setString method to add data to an SQLXML object.	SQLXML	SQLException	Yes
createStatement()	Creates a statement object that sends SQL statements to a database.	Statement	SQLException	Yes
createStatement(int resultSetType, int resultSetConcurrency)	Creates a statement object that will generate a ResultSet object with a given type and concurrency.	Statement	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a statement object that will generate a ResultSet object with a given type and concurrency.	Statement	SQLException	Yes
getAutoCommit()	Retrieves the current autocommit mode for this connection object.	boolean	SQLException	Yes
getCatalog()	Obtains the current directory name of this connection object.	String	SQLException	Yes
getClientInfo()	Returns a list containing the name and current value of each client information property supported by the driver. If the client information property has not been set and has no default value, the value of this property may be null .	Properties	SQLException	Yes
getClientInfo(String name)	Returns the value of the client information property specified by name. This method may return null if the specified client information property has not been set and there is no default value. This method also returns null if the driver does not support the specified client information property name.	String	SQLException	Yes
getHoldability()	Obtains the current holdability of the ResultSet object created using this connection object.	int	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
getMetaData()	Retrieves the DatabaseMetaData object that contains metadata about the database to which this connection object represents the connection. Metadata includes information about database tables, supported SQL syntax, stored procedures, and functions of the connection.	DatabaseMetaData	SQLException	Yes
getNetworkTimeout()	Obtains the number of milliseconds the driver will wait for a database request to complete. If the limit is exceeded, SQLException is thrown.	int	SQLException	Yes
getSchema()	Retrieves the current schema name of this connection object.	String	SQLException	Yes
getTransactionIsolation()	Retrieves the current transaction isolation level of this connection object.	int	SQLException	Yes
getTypeMap()	Retrieves the map object associated with this connection object. Unless the application adds an entry, the returned type mapping will be empty.	Map<String,Class<?>>	SQLException	Yes
getWarnings()	Retrieves the first warning reported by calls on this connection object. If there are multiple warnings, subsequent warnings are linked to the first warning and can be retrieved by calling the SQLWarning.getNextWarning method on the previously retrieved warning.	SQLWarning	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
isClosed()	Checks whether this connection object has been closed. If a method has been called to close the connection, or if some fatal error occurs, the connection is closed. Ensure that true is returned only when this method is called after the close ction.Close method is called.	boolean	SQLException	Yes
isReadOnly()	Checks whether this connection object is in read-only mode.	boolean	SQLException	Yes
isValid(int timeout)	Returns true if the connection has not been closed and is still valid. The driver should commit a query to the connection, or use other mechanisms to affirmatively verify that the connection is still valid when this method is called.	Boolean	SQLException	Yes
nativeSQL(String sql)	Converts a given SQL statement to the native SQL syntax of the system. The driver can convert the JDBC SQL syntax to the native SQL syntax of its system before sending it. This method returns the native form of the statement that the driver should have sent.	String	SQLException	Yes
prepareCall(String sql)	Creates a CallableStatement object that calls a database stored procedure. The CallableStatement object provides methods for setting its IN and OUT parameters, as well as methods for executing calls to stored procedures.	CallableStatement	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	Creates a CallableStatement object. This object will generate a ResultSet object with the given type and concurrency. This method is the same as the preparation call method above, but it allows the default result set type and concurrency to be overwritten.	CallableStatement	SQLException	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a CallableStatement object. This object will generate a ResultSet object with the given type and concurrency. This method is the same as the preparation call method above, but it allows the default result set type, result set concurrency type, and holdability to be overwritten.	CallableStatement	SQLException	Yes
prepareStatement(String sql)	Creates a prepared statement object for sending parameterized SQL statements to a database.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int autoGeneratedKeys)	Creates a default prepared statement object that can retrieve automatically generated keys. The given constant tells the driver whether it should make the automatically generated key available for retrieval. If the SQL statement is not an INSERT statement or an SQL statement that can return an automatically generated key, this parameter is ignored.	PreparedStatement	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
prepareStatement(String sql, int[] columnIndexes)	Creates a default prepared statement object that can return an automatically generated key specified by a given array. This array contains the indexes of the columns in the target table that contain the automatically generated keys, which should be available. If the SQL statement is not an INSERT statement or an SQL statement that can return an automatically generated key, the driver ignores the array.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	Creates a prepared statement object that will generate a ResultSet object with a given type and concurrency. This method is the same as the prepared statement method above, but it allows the default result set type and concurrency to be overwritten.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a prepared statement object that will generate a ResultSet object with a given type, concurrency, and holdability.	PreparedStatement	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
prepareStatement(String sql, String[] columnNames)	Creates a default prepared statement object that can return an automatically generated key specified by a given array. This array contains the names of the columns in the target table that contain the automatically generated keys, which should be returned. If the SQL statement is not an INSERT statement or an SQL statement that can return an automatically generated key, the driver ignores the array.	PreparedStatement	SQLException	Yes
releaseSavepoint(Savepoint savepoint)	Deletes the specified savepoint and subsequent savepoint objects from the current transaction. Any reference to the savepoint after it is deleted will cause an SQL exception to be thrown.	void	SQLException	Yes
rollback()	Undoes all changes made in the current transaction and releases any database locks currently held by this connection object. This method should be used only when the autocommit mode is disabled.	void	SQLException	Yes
rollback(Savepoint savepoint)	Undoes all changes made after a given savepoint object is set. This method should be used only when the autocommit mode is disabled.	void	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
setAutoCommit(boolean autoCommit)	Sets the autocommit mode of this connection to the given state. If a connection is in autocommit mode, all its SQL statements are executed and committed as a single transaction. Otherwise, its SQL statements are grouped into transactions that are terminated by calls to method commits or method rollbacks. By default, new connections are in autocommit mode.	void	SQLException	Yes
setClientInfo(Properties properties)	Sets the client information property value of the connection. The properties object contains the names and values of the client information properties to be set. The client information property set contained in the attribute list replaces the current client information property set on the connection. If the property currently set on the connection is not in the property list, the property is cleared.	void	SQLException	Yes
setClientInfo(String name,String value)	Sets the client info property value specified by name to that specified by value .	void	SQLException	Yes
setHoldability(int holdability)	Changes the default holdability of the ResultSet object created with this connection object to the given holdability. The default holdability of the ResultSet object can be determined by calling DatabaseMetaData.getResultSetHoldability.	void	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
setNetworkTimeout(Executor executor, int milliseconds)	Sets the maximum time that a connection or an object created from a connection waits for the database to reply to any request. If any request is still not answered, the waiting method returns SQLException, and the connection or object created from the connection is marked as closed. Any subsequent use of the object other than the Close, isCabled, or Connection.isValid method will result in SQLException.	void	SQLException	Yes
setReadOnly(boolean readOnly)	Sets this connection to read-only mode as a hint for the driver to enable database optimization.	void	SQLException	Yes
setSavepoint()	Creates an unnamed savepoint in the current transaction and returns a new savepoint object representing it.	Savepoint	SQLException	Yes
setSavepoint(String name)	Creates a savepoint with the given name in the current transaction and returns a new savepoint object representing it.	Savepoint	SQLException	Yes
setSchema(String schema)	Sets the given schema name to access.	void	SQLException	Yes
setTransactionIsolation(int level)	Attempts to change the transaction isolation level of this connection object to a given level. Constants defined in the connection API are possible transaction isolation levels.	void	SQLException	Yes

Method	Description	Return Type	throws	Support JDBC 4
setTypeMap(Map<String,Class<?>> map)	Installs the given TypeMap object as the type mapping for this connection object. Type mappings will be used for SQL structured types and custom mappings for different types.	void	SQLException	Yes

 NOTE

- The autocommit mode is used by default within the API. If you disable it by running **setAutoCommit(false)**, all the statements executed later will be packaged in explicit transactions, and you cannot execute statements that cannot be executed within transactions.
- When the PreparedStatement class is created, using the returning clause in the SQL statement does not take effect.

5.4.4.2 java.sql.CallableStatement

java.sql.CallableStatement is an API for executing stored procedures.

Table 5-15 Support status for java.sql.CallableStatement

Method Name	Return Type	JDBC4 Supported or Not
getArray(int parameterIndex)	Array	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBlob(int parameterIndex)	Blob	Yes
getBoolean(int parameterIndex)	Boolean	Yes
getByte(int parameterIndex)	byte	Yes
getBytes(int parameterIndex)	byte[]	Yes
getClob(int parameterIndex)	Clob	Yes
getDate(int parameterIndex)	Date	Yes
getDate(int parameterIndex, Calendar cal)	Date	Yes
getDouble(int parameterIndex)	double	Yes

Method Name	Return Type	JDBC4 Supported or Not
getFloat(int parameterIndex)	float	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getObject(int parameterIndex)	Object	Yes
getObject(int parameterIndex, Class<T> type)	Object	Yes
getShort(int parameterIndex)	short	Yes
getSQLXML(int parameterIndex)	SQLXML	Yes
getString(int parameterIndex)	String	Yes
getNString(int parameterIndex)	String	Yes
getTime(int parameterIndex)	Time	Yes
getTime(int parameterIndex, Calendar cal)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getTimestamp(int parameterIndex, Calendar cal)	Timestamp	Yes
registerOutParameter(int parameterIndex, int type)	void	Yes
registerOutParameter(int parameterIndex, int sqlType, int type)	void	Yes
wasNull()	Boolean	Yes

 **NOTE**

- The batch operation of statements containing OUT parameter is not allowed.
- The following methods are inherited from java.sql.Statement: close, execute, executeQuery, executeUpdate, getConnection, getResultSet, getUpdateCount, isClosed, setMaxRows, and setFetchSize.
- The following methods are inherited from java.sql.PreparedStatement: addBatch, clearParameters, execute, executeQuery, executeUpdate, getMetaData, setBigDecimal, setBoolean, setByte, setBytes, setDate, setDouble, setFloat, setInt, setLong, setNull, setObject, setString, setTime, and setTimestamp.
- The registerOutParameter(int parameterIndex, int sqlType, int type) method is used only to register the composite data type.

5.4.4.3 java.sql.DatabaseMetaData

java.sql.DatabaseMetaData is an API for defining database objects.

Table 5-16 Support status for java.sql.DatabaseMetaData

Method	Return Type	Support JDBC 4
allProceduresAreCallable()	Boolean	Yes
allTablesAreSelectable()	Boolean	Yes
autoCommitFailureClosesAllResultSets()	Boolean	Yes
dataDefinitionCausesTransactionCommit()	Boolean	Yes
dataDefinitionIgnoredInTransactions()	Boolean	Yes
deletesAreDetected(int type)	Boolean	Yes
doesMaxRowSizeIncludeBlobs()	Boolean	Yes
generatedKeyAlwaysReturned()	Boolean	Yes
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	ResultSet	Yes
getCatalogs()	ResultSet	Yes
getCatalogSeparator()	String	Yes
getCatalogTerm()	String	Yes
getClientInfoProperties()	ResultSet	Yes

Method	Return Type	Support JDBC 4
getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	ResultSet	Yes
getConnection()	Connection	Yes
getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	ResultSet	Yes
getDefaultTransactionIsolation()	int	Yes
getExportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getExtraNameCharacters()	String	Yes
getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	ResultSet	Yes
getFunctions(String catalog, String schemaPattern, String functionNamePattern)	ResultSet	Yes
getIdentifierQuoteString()	String	Yes
getImportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	ResultSet	Yes
getMaxBinaryLiteralLength()	int	Yes
getMaxCatalogNameLength()	int	Yes
getMaxCharLiteralLength()	int	Yes
getMaxColumnNameLength()	int	Yes

Method	Return Type	Support JDBC 4
getMaxColumnsInGroupBy()	int	Yes
getMaxColumnsInIndex()	int	Yes
getMaxColumnsInOrderBy()	int	Yes
getMaxColumnsInSelect()	int	Yes
getMaxColumnsInTable()	int	Yes
getMaxConnections()	int	Yes
getMaxCursorNameLength()	int	Yes
getMaxIndexLength()	int	Yes
getMaxLogicalLobSize()	default long	Yes
getMaxProcedureName- Length()	int	Yes
getMaxRowSize()	int	Yes
getMaxSchemaName- Length()	int	Yes
getMaxStatementLength()	int	Yes
getMaxStatements()	int	Yes
getMaxTableNameLength()	int	Yes
getMaxTablesInSelect()	int	Yes
getMaxUserNameLength()	int	Yes
getNumericFunctions()	String	Yes
getPrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getPartitionTablePrimary- Keys(String catalog, String schema, String table)	ResultSet	Yes
getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	ResultSet	Yes
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	ResultSet	Yes

Method	Return Type	Support JDBC 4
getProcedureTerm()	String	Yes
getSchemas()	ResultSet	Yes
getSchemas(String catalog, String schemaPattern)	ResultSet	Yes
getSchemaTerm()	String	Yes
getSearchStringEscape()	String	Yes
getSQLKeywords()	String	Yes
getSQLStateType()	int	Yes
getStringFunctions()	String	Yes
getSystemFunctions()	String	Yes
getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	ResultSet	Yes
getTimeDateFunctions()	String	Yes
getTypeInfo()	ResultSet	Yes
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	ResultSet	Yes
getURL()	String	Yes
getVersionColumns(String catalog, String schema, String table)	ResultSet	Yes
insertsAreDetected(int type)	Boolean	Yes
locatorsUpdateCopy()	Boolean	Yes
othersDeletesAreVisible(int type)	Boolean	Yes
othersInsertsAreVisible(int type)	Boolean	Yes
othersUpdatesAreVisible(int type)	Boolean	Yes
ownDeletesAreVisible(int type)	Boolean	Yes
ownInsertsAreVisible(int type)	Boolean	Yes

Method	Return Type	Support JDBC 4
ownUpdatesAreVisible(int type)	Boolean	Yes
storesLowerCaselIdentifiers()	Boolean	Yes
storesMixedCaselIdentifiers()	Boolean	Yes
storesUpperCaselIdentifiers()	Boolean	Yes
supportsBatchUpdates()	Boolean	Yes
supportsCatalogsInDataManipulation()	Boolean	Yes
supportsCatalogsInIndexDefinitions()	Boolean	Yes
supportsCatalogsInPrivilegeDefinitions()	Boolean	Yes
supportsCatalogsInProcedureCalls()	Boolean	Yes
supportsCatalogsInTableDefinitions()	Boolean	Yes
supportsCorrelatedSubqueries()	Boolean	Yes
supportsDataDefinitionAndDataManipulationTransactions()	Boolean	Yes
supportsDataManipulationTransactionsOnly()	Boolean	Yes
supportsGetGeneratedKeys()	Boolean	Yes
supportsMixedCaselIdentifiers()	Boolean	Yes
supportsMultipleOpenResults()	Boolean	Yes
supportsNamedParameters()	Boolean	Yes
supportsOpenCursorsAcrossCommit()	Boolean	Yes
supportsOpenCursorsAcrossRollback()	Boolean	Yes
supportsOpenStatementsAcrossCommit()	Boolean	Yes
supportsOpenStatementsAcrossRollback()	Boolean	Yes

Method	Return Type	Support JDBC 4
supportsPositionedDelete()	Boolean	Yes
supportsPositionedUpdate()	Boolean	Yes
supportsRefCursors()	Boolean	Yes
supportsResultSetConcurrency(int type, int concurrency)	Boolean	Yes
supportsResultSetType(int type)	Boolean	Yes
supportsSchemasInIndexDefinitions()	Boolean	Yes
supportsSchemasInPrivilegeDefinitions()	Boolean	Yes
supportsSchemasInProcedureCalls()	Boolean	Yes
supportsSchemasInTableDefinitions()	Boolean	Yes
supportsSelectForUpdate()	Boolean	Yes
supportsStatementPooling()	Boolean	Yes
supportsStoredFunctionsUsingCallSyntax()	Boolean	Yes
supportsStoredProcedures()	Boolean	Yes
supportsSubqueriesInComparisons()	Boolean	Yes
supportsSubqueriesInExists()	Boolean	Yes
supportsSubqueriesInIns()	Boolean	Yes
supportsSubqueriesInQuantifieds()	Boolean	Yes
supportsTransactionIsolationLevel(int level)	Boolean	Yes
supportsTransactions()	Boolean	Yes
supportsUnion()	Boolean	Yes
supportsUnionAll()	Boolean	Yes
updatesAreDetected(int type)	Boolean	Yes

Method	Return Type	Support JDBC 4
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	ResultSet	Yes
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	ResultSet	Yes
getTableTypes()	ResultSet	Yes
getUserName()	String	Yes
isReadOnly()	Boolean	Yes
nullsAreSortedHigh()	Boolean	Yes
nullsAreSortedLow()	Boolean	Yes
nullsAreSortedAtStart()	Boolean	Yes
nullsAreSortedAtEnd()	Boolean	Yes
getDatabaseProductName()	String	Yes
getDatabaseProductVersion()	String	Yes
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes
getDriverMinorVersion()	int	Yes
usesLocalFiles()	Boolean	Yes
usesLocalFilePerTable()	Boolean	Yes
supportsMixedCaseIdentifiers()	Boolean	Yes
storesUpperCaseIdentifiers()	Boolean	Yes
storesLowerCaseIdentifiers()	Boolean	Yes
supportsMixedCaseQuotedIdentifiers()	Boolean	Yes
storesUpperCaseQuotedIdentifiers()	Boolean	Yes
storesLowerCaseQuotedIdentifiers()	Boolean	Yes

Method	Return Type	Support JDBC 4
storesMixedCaseQuotedIdentifiers()	Boolean	Yes
supportsAlterTableWithAddColumn()	Boolean	Yes
supportsAlterTableWithDropColumn()	Boolean	Yes
supportsColumnAliasing()	Boolean	Yes
nullPlusNonNullIsNull()	Boolean	Yes
supportsConvert()	Boolean	Yes
supportsConvert(int fromType, int toType)	Boolean	Yes
supportsTableCorrelationNames()	Boolean	Yes
supportsDifferentTableCorrelationNames()	Boolean	Yes
supportsExpressionsInOrderBy()	Boolean	Yes
supportsOrderByUnrelated()	Boolean	Yes
supportsGroupBy()	Boolean	Yes
supportsGroupByUnrelated()	Boolean	Yes
supportsGroupByBeyondSelect()	Boolean	Yes
supportsLikeEscapeClause()	Boolean	Yes
supportsMultipleResultSets()	Boolean	Yes
supportsMultipleTransactions()	Boolean	Yes
supportsNonNullableColumns()	Boolean	Yes
supportsMinimumSQLGrammar()	Boolean	Yes
supportsCoreSQLGrammar()	Boolean	Yes
supportsExtendedSQLGrammar()	Boolean	Yes
supportsANSI92EntryLevelSQL()	Boolean	Yes

Method	Return Type	Support JDBC 4
supportsANSI92IntermediateSQL()	Boolean	Yes
supportsANSI92FullSQL()	Boolean	Yes
supportsIntegrityEnhancementFacility()	Boolean	Yes
supportsOuterJoins()	Boolean	Yes
supportsFullOuterJoins()	Boolean	Yes
supportsLimitedOuterJoins()	Boolean	Yes
isCatalogAtStart()	Boolean	Yes
supportsSchemasInDataManipulation()	Boolean	Yes
supportsSavepoints()	Boolean	Yes
supportsResultSetHoldability(int holdability)	Boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes
getDatabaseMinorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes

 **NOTE**

The `getPartitionTablePrimaryKeys(String catalog, String schema, String table)` method is used to obtain the primary key column of a partitioned table that contains global indexes. The following is an example:

```
PgDatabaseMetaData dbmd = (PgDatabaseMetaData)conn.getMetaData();
dbmd.getPartitionTablePrimaryKeys("catalogName", "schemaName", "tableName");
```

5.4.4.4 java.sql.Driver

java.sql.Driver is a database driver API.

Table 5-17 Support status for java.sql.Driver

Method Name	Return Type	JDBC4 Supported or Not
acceptsURL(String url)	Boolean	Yes
connect(String url, Properties info)	Connection	Yes

Method Name	Return Type	JDBC4 Supported or Not
jdbcCompliant()	Boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes
getParentLogger()	Logger	Yes
getPropertyInfo(String url, Properties info)	DriverPropertyInfo[]	Yes

5.4.4.5 java.sql.PreparedStatement

java.sql.PreparedStatement is a prepared statement API.

Table 5-18 Support status for java.sql.PreparedStatement

Method Name	Return Type	JDBC4 Supported or Not
clearParameters()	void	Yes
execute()	Boolean	Yes
executeQuery()	ResultSet	Yes
excuteUpdate()	int	Yes
executeLargeUpdate()	long	No
getMetaData()	ResultSetMetaData	Yes
getParameterMetaData()	ParameterMetaData	Yes
setArray(int parameterIndex, Array x)	void	Yes
setAsciiStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, int length)	void	Yes

Method Name	Return Type	JDBC4 Supported or Not
setBinaryStream(int parameterIndex, InputStream x, long length)	void	Yes
setBlob(int parameterIndex, InputStream inputStream)	void	Yes
setBlob(int parameterIndex, InputStream inputStream, long length)	void	Yes
setBlob(int parameterIndex, Blob x)	void	Yes
setCharacterStream(int parameterIndex, Reader reader)	void	Yes
setCharacterStream(int parameterIndex, Reader reader, int length)	void	Yes
setClob(int parameterIndex, Reader reader)	void	Yes
setClob(int parameterIndex, Reader reader, long length)	void	Yes
setClob(int parameterIndex, Clob x)	void	Yes
setDate(int parameterIndex, Date x, Calendar cal)	void	Yes
setNull(int parameterIndex, int sqlType)	void	Yes
setNull(int parameterIndex, int sqlType, String typeName)	void	Yes

Method Name	Return Type	JDBC4 Supported or Not
setObject(int parameterIndex, Object x)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	void	Yes
setSQLXML(int parameterIndex, SQLXML xmlObject)	void	Yes
setTime(int parameterIndex, Time x)	void	Yes
setTime(int parameterIndex, Time x, Calendar cal)	void	Yes
setTimestamp(int parameterIndex, Timestamp x)	void	Yes
setTimestamp(int parameterIndex, Timestamp x, Calendar cal)	void	Yes
setUnicodeStream(int parameterIndex, InputStream x, int length)	void	Yes
setURL(int parameterIndex, URL x)	void	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	void	Yes
setByte(int parameterIndex, byte x)	void	Yes

Method Name	Return Type	JDBC4 Supported or Not
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes
setDouble(int parameterIndex, double x)	void	Yes
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes
setNString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes

 **NOTE**

- Execute addBatch() and execute() only after executing clearBatch().
- Batch is not cleared by calling executeBatch(). Clear batch by explicitly calling clearBatch().
- After bound variables of a batch are added, if you want to reuse these values, you do not need to use set*() again. Instead, add a batch.
- The following methods are inherited from java.sql.Statement: close(), execute(), executeQuery(), executeUpdate(), getConnection(), getResultSet(), getUpdateCount(), isClosed(), setMaxRows(), and setFetchSize ().
- The executeLargeUpdate() method can only be used in JDBC 4.2 or later.

5.4.4.6 java.sql.ResultSet

java.sql.ResultSet is an execution result set API.

Table 5-19 Support status for java.sql.ResultSet

Method Name	Return Type	JDBC4 Supported or Not
absolute(int row)	Boolean	Yes
afterLast()	void	Yes
beforeFirst()	void	Yes
cancelRowUpdates()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
deleteRow()	void	Yes
findColumn(String columnLabel)	int	Yes
first()	Boolean	Yes
getArray(int columnIndex)	Array	Yes
getArray(String columnLabel)	Array	Yes
getAsciiStream(int columnIndex)	InputStream	Yes
getAsciiStream(String columnLabel)	InputStream	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	Yes
getBinaryStream(int columnIndex)	InputStream	Yes
getBinaryStream(String columnLabel)	InputStream	Yes
getBlob(int columnIndex)	Blob	Yes
getBlob(String columnLabel)	Blob	Yes
getBoolean(int columnIndex)	Boolean	Yes
getBoolean(String columnLabel)	Boolean	Yes
getByte(int columnIndex)	byte	Yes

Method Name	Return Type	JDBC4 Supported or Not
getBytes(int columnIndex)	byte[]	Yes
getBytes(String columnLabel)	byte	Yes
getBytes(String columnLabel)	byte[]	Yes
getCharacterStream(int columnIndex)	Reader	Yes
getCharacterStream(String columnLabel)	Reader	Yes
getClob(int columnIndex)	Clob	Yes
getClob(String columnLabel)	Clob	Yes
getConcurrency()	int	Yes
getCursorName()	String	Yes
getDate(int columnIndex)	Date	Yes
getDate(int columnIndex, Calendar cal)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDate(String columnLabel, Calendar cal)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getFloat(int columnIndex)	float	Yes
getFloat(String columnLabel)	float	Yes
getInt(int columnIndex)	int	Yes
getInt(String columnLabel)	int	Yes

Method Name	Return Type	JDBC4 Supported or Not
getLong(int columnIndex)	long	Yes
getLong(String columnLabel)	long	Yes
getMetaData()	ResultSetMetaData	Yes
getObject(int columnIndex)	Object	Yes
getObject(int columnIndex, Class<T> type)	<T> T	Yes
getObject(int columnIndex, Map<String,Class<?>> map)	Object	Yes
getObject(String columnLabel)	Object	Yes
getObject(String columnLabel, Class<T> type)	<T> T	Yes
getObject(String columnLabel, Map<String,Class<?>> map)	Object	Yes
getRow()	int	Yes
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes
getSQLXML(int columnIndex)	SQLXML	Yes
getSQLXML(String columnLabel)	SQLXML	Yes
getStatement()	Statement	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getNString(int columnIndex)	String	Yes

Method Name	Return Type	JDBC4 Supported or Not
getNString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(int columnIndex, Calendar cal)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTime(String columnLabel, Calendar cal)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(int columnIndex, Calendar cal)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
getTimestamp(String columnLabel, Calendar cal)	Timestamp	Yes
getType()	int	Yes
getWarnings()	SQLWarning	Yes
insertRow()	void	Yes
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes
isClosed()	Boolean	Yes
isFirst()	Boolean	Yes
isLast()	Boolean	Yes
last()	Boolean	Yes
moveToCurrentRow()	void	Yes
moveToInsertRow()	void	Yes
next()	Boolean	Yes
previous()	Boolean	Yes
refreshRow()	void	Yes

Method Name	Return Type	JDBC4 Supported or Not
relative(int rows)	Boolean	Yes
rowDeleted()	Boolean	Yes
rowInserted()	Boolean	Yes
rowUpdated()	Boolean	Yes
setFetchDirection(int direction)	void	Yes
setFetchSize(int rows)	void	Yes
updateArray(int columnIndex, Array x)	void	Yes
updateArray(String columnLabel, Array x)	void	Yes
updateAsciiStream(int columnIndex, InputStream x, int length)	void	Yes
updateAsciiStream(String columnLabel, InputStream x, int length)	void	Yes
updateBigDecimal(int columnIndex, BigDecimal x)	void	Yes
updateBigDecimal(String columnLabel, BigDecimal x)	void	Yes
updateBinaryStream(int columnIndex, InputStream x, int length)	void	Yes
updateBinaryStream(String columnLabel, InputStream x, int length)	void	Yes
updateBoolean(int columnIndex, boolean x)	void	Yes
updateBoolean(String columnLabel, boolean x)	void	Yes
updateByte(int columnIndex, byte x)	void	Yes
updateByte(String columnLabel, byte x)	void	Yes

Method Name	Return Type	JDBC4 Supported or Not
updateBytes(int columnIndex, byte[] x)	void	Yes
updateBytes(String columnLabel, byte[] x)	void	Yes
updateCharacter-Stream(int columnIndex, Reader x, int length)	void	Yes
updateCharacter-Stream(String columnLabel, Reader reader, int length)	void	Yes
updateDate(int columnIndex, Date x)	void	Yes
updateDate(String columnLabel, Date x)	void	Yes
updateDouble(int columnIndex, double x)	void	Yes
updateDouble(String columnLabel, double x)	void	Yes
updateFloat(int columnIndex, float x)	void	Yes
updateFloat(String columnLabel, float x)	void	Yes
updateInt(int columnIndex, int x)	void	Yes
updateInt(String columnLabel, int x)	void	Yes
updateLong(int columnIndex, long x)	void	Yes
updateLong(String columnLabel, long x)	void	Yes
updateNull(int columnIndex)	void	Yes
updateNull(String columnLabel)	void	Yes
updateObject(int columnIndex, Object x)	void	Yes

Method Name	Return Type	JDBC4 Supported or Not
updateObject(int columnIndex, Object x, int scaleOrLength)	void	Yes
updateObject(String columnLabel, Object x)	void	Yes
updateObject(String columnLabel, Object x, int scaleOrLength)	void	Yes
updateRow()	void	Yes
updateShort(int columnIndex, short x)	void	Yes
updateShort(String columnLabel, short x)	void	Yes
updateSQLXML(int columnIndex, SQLXML xmlObject)	void	Yes
updateSQLXML(String columnLabel, SQLXML xmlObject)	void	Yes
updateString(int columnIndex, String x)	void	Yes
updateString(String columnLabel, String x)	void	Yes
updateTime(int columnIndex, Time x)	void	Yes
updateTime(String columnLabel, Time x)	void	Yes
updateTimestamp(int columnIndex, Timestamp x)	void	Yes
updateTimestamp(String columnLabel, Timestamp x)	void	Yes
wasNull()	Boolean	Yes

 NOTE

- One statement cannot have multiple open ResultSets.
- The cursor that is used for traversing the ResultSet cannot be open after being committed.

5.4.4.7 java.sql.ResultSetMetaData

java.sql.ResultSetMetaData describes the information about ResultSet objects.

Table 5-20 Support status for java.sql.ResultSetMetaData

Method Name	Return Type	JDBC4 Supported or Not
getCatalogName(int column)	String	Yes
getColumnClassName(int column)	String	Yes
getColumnCount()	int	Yes
getColumnDisplaySize(int column)	int	Yes
getColumnLabel(int column)	String	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes
getPrecision(int column)	int	Yes
getScale(int column)	int	Yes
getSchemaName(int column)	String	Yes
getTableName(int column)	String	Yes
isAutoIncrement(int column)	Boolean	Yes
isCaseSensitive(int column)	Boolean	Yes
isCurrency(int column)	Boolean	Yes
isDefinitelyWritable(int column)	Boolean	Yes
isNullable(int column)	int	Yes
isReadOnly(int column)	Boolean	Yes
isSearchable(int column)	Boolean	Yes

Method Name	Return Type	JDBC4 Supported or Not
isSigned(int column)	Boolean	Yes
isWritable(int column)	Boolean	Yes

5.4.4.8 java.sql.Statement

java.sql.Statement is an SQL statement API.

Table 5-21 Support status for java.sql.Statement

Method Name	Return Type	JDBC4 Supported or Not
addBatch(String sql)	void	Yes
clearBatch()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
closeOnCompletion()	void	Yes
execute(String sql)	Boolean	Yes
execute(String sql, int autoGeneratedKeys)	Boolean	Yes
execute(String sql, int[] columnIndexes)	Boolean	Yes
execute(String sql, String[] columnNames)	Boolean	Yes
executeBatch()	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
executeUpdate(String sql, int autoGeneratedKeys)	int	Yes
executeUpdate(String sql, int[] columnIndexes)	int	Yes

Method Name	Return Type	JDBC4 Supported or Not
executeUpdate(String sql, String[] columnNames)	int	Yes
getConnection()	Connection	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getGeneratedKeys()	ResultSet	Yes
getMaxFieldSize()	int	Yes
getMaxRows()	int	Yes
getMoreResults()	Boolean	Yes
getMoreResults(int current)	Boolean	Yes
getResultSet()	ResultSet	Yes
getResultSetConcurrency()	int	Yes
getResultSetHoldability()	int	Yes
getResultSetType()	int	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isCloseOnCompletion()	Boolean	Yes
isPoolable()	Boolean	Yes
setCursorName(String name)	void	Yes
setEscapeProcessing(boolean enable)	void	Yes
setFetchDirection(int direction)	void	Yes
setMaxFieldSize(int max)	void	Yes
setMaxRows(int max)	void	Yes

Method Name	Return Type	JDBC4 Supported or Not
setPoolable(boolean poolable)	void	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes
executeLargeUpdate(String sql)	long	No
getLargeUpdateCount()	long	No
executeLargeBatch()	long	No
executeLargeUpdate(String sql, int autoGeneratedKeys)	long	No
executeLargeUpdate(String sql, int[] columnIndexes)	long	No
executeLargeUpdate(String sql, String[] columnNames)	long	No

 NOTE

- Using setFetchSize can reduce the memory occupied by result sets on the client. Result sets are packaged into cursors and segmented for processing, which will increase the communication traffic between the database and the client, affecting performance.
- Database cursors are valid only within their transactions. If **setFetchSize** is set, set **setAutoCommit(false)** and commit transactions on the connection to flush service data to a database.
- The LargeUpdate method can only be used in JDBC 4.2 or later.

5.4.4.9 javax.sql.ConnectionPoolDataSource

This section describes **javax.sql.ConnectionPoolDataSource**, the interface for data source connection pools.

Table 5-22 Support status for javax.sql.ConnectionPoolDataSource

Method Name	Return Type	Support JDBC 4
getPooledConnection()	PooledConnection	Yes

Method Name	Return Type	Support JDBC 4
getPooledConnection(String user,String password)	PooledConnection	Yes

5.4.4.10 javax.sql.DataSource

This section describes **javax.sql.DataSource**, the interface for data sources.

Table 5-23 Support status for javax.sql.DataSource

Method Name	Return Type	Support JDBC 4
getConnection()	Connection	Yes
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

5.4.4.11 javax.sql.PooledConnection

This section describes **javax.sql.PooledConnection**, the connection interface created by a connection pool.

Table 5-24 Support status for javax.sql.PooledConnection

Method Name	Return Type	Support JDBC 4
addConnectionEventListener(ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes
removeConnectionEventListener(ConnectionEventListener listener)	void	Yes

5.4.4.12 javax.naming.Context

This section describes **javax.naming.Context**, the context interface for connection configuration.

Table 5-25 Support status for javax.naming.Context

Method Name	Return Type	Support JDBC 4
bind(Name name, Object obj)	void	Yes
bind(String name, Object obj)	void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes
rebind(Name name, Object obj)	void	Yes
rebind(String name, Object obj)	void	Yes
rename(Name oldName, Name newName)	void	Yes
rename(String oldName, String newName)	void	Yes
unbind(Name name)	void	Yes
unbind(String name)	void	Yes

5.4.4.13 javax.naming.spi.InitialContextFactory

This section describes **javax.naming.spi.InitialContextFactory**, the initial context factory interface.

Table 5-26 Support status for javax.naming.spi.InitialContextFactory

Method Name	Return Type	Support JDBC 4
getInitialContext(Hashtable<?,?> environment)	Context	Yes

5.4.4.14 CopyManager

CopyManager is an API class provided by the JDBC driver in GaussDB. It is used to import data to GaussDB clusters in batches.

Inheritance Relationship of CopyManager

The CopyManager class is in the **com.huawei.opengauss.jdbc.copy** package and inherits the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```

Construction Method

```
public CopyManager(BaseConnection connection)
throws SQLException
```

Common Methods

Table 5-27 Common methods of CopyManager

Method Name	Return Type	Description	throws	Support JDBC 4.0 (Yes/No)
copyIn(String sql)	Copy In	-	SQLException	Yes
copyIn(String sql, InputStream from)	long	Uses COPY FROM STDIN to quickly load data to tables in the database from InputStream.	SQLException,IOException	Yes
copyIn(String sql, InputStream from, int bufferSize)	long	Uses COPY FROM STDIN to quickly load data to tables in the database from InputStream.	SQLException,IOException	Yes
copyIn(String sql, Reader from)	long	Uses COPY FROM STDIN to quickly load data to tables in the database from Reader.	SQLException,IOException	Yes
copyIn(String sql, Reader from, int bufferSize)	long	Uses COPY FROM STDIN to quickly load data to tables in the database from Reader.	SQLException,IOException	Yes
copyOut(String sql)	Copy Out	-	SQLException	Yes
copyOut(String sql, OutputStream to)	long	Sends the result set of COPY TO STDOUT from the database to the OutputStream class.	SQLException,IOException	Yes
copyOut(String sql, Writer to)	long	Sends the result set of COPY TO STDOUT from the database to the Writer class.	SQLException,IOException	Yes

5.4.4.15 PGReplicationConnection

PGReplicationConnection is an API class provided by the JDBC driver in GaussDB. It is used to implement functions related to logical replication.

Inheritance Relationship of PGReplicationConnection

PGReplicationConnection is a logical replication API. Its implementation class is PGReplicationConnectionImpl, which is in the **com.huawei.opengauss.jdbc.replication** package. The declaration of the class is as follows:

```
public class PGReplicationConnection implements PGReplicationConnection
```

Constructor Method

```
public PGReplicationConnection(BaseConnection connection)
```

Common Methods

Table 5-28 Common methods of PGReplicationConnection

Method Name	Return Type	Description	Throws
createReplicationSlot()	ChainedCreateReplicationSlotBuilder	Creates a logical replication slot. Only cluster-level (CSN-based) logical replication slots can be created for connecting to CNs, and replication slots with the same name can be created on other CNs and primary DN. Only local (LSN-based) logical replication slots can be created for connecting to DNs. For details about how to create an LSN-based logical replication slot on a CN or a CSN-based logical replication slot on a DN, see the SQL function <code>pg_create_logical_replication_slot</code> for logical replication.	-

Method Name	Return Type	Description	Throws
dropReplicationSlot(String slotName)	void	Deletes a logical replication slot. When you connect to a CN to delete a logical replication slot, if the logical replication slot is an LSN-based logical replication slot, only the replication slot of the current node is deleted. Replication slots with the same name on other nodes are not affected. When a CSN-based logical replication slot with the same name exists on other nodes, no error is reported because some nodes do not have replication slots. In addition, replication slots with the same name on all nodes are successfully deleted. If no replication slot exists on any node, an error is reported. If an LSN-based logical replication slot remains on the current CN and a CSN-based logical replication slot with the same name remains on other nodes, connecting to a CN to delete a replication slot will delete only the local LSN-based logical replication slot. After the deletion is complete, perform the deletion operation again to delete the replication slots with the same name on other nodes.	SQLException,IO Exception
replicationStream()	ChainedStreamBuilder	Enables logical replication.	-

5.4.4.16 PGReplicationStream

PGReplicationStream is an API class provided by the GaussDB JDBC driver. It is used to operate logical replication streams.

Inheritance Relationship of PGReplicationStream

PGReplicationStream is a logical replication API. Its implementation class is V3PGReplicationStream, which is in the

com.huawei.opengauss.jdbc.core.v3.replication package. The declaration of the class is as follows:

```
public class V3PGReplicationStream implements PGReplicationStream
```

Constructor Method

```
public V3PGReplicationStream(CopyDual copyDual, LogSequenceNumber  
startLSN, long updateIntervalMs, ReplicationType replicationType)
```

Common Methods

Table 5-29 Common methods of PGReplicationConnection

Method Name	Return Type	Description	throws
close()	void	Ends the logical replication and releases resources.	SQLException
forceUpdateStatus()	void	Forcibly sends the LSN status received, flushed, and applied last time to the backend.	SQLException
getLastAppliedLSN()	LogSequenceNumber	Obtains the LSN when the primary node replays logs last time.	-
getLastFlushedLSN()	LogSequenceNumber	Obtains the LSN flushed by the primary node last time, that is, the LSN pushed by the current logical decoding.	-
getLastReceivedLSN()	LogSequenceNumber	Obtains the last received LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots).	-
isClosed()	boolean	Determines whether the replication stream is disabled.	-
read()	ByteBuffer	Reads the next WAL record from the backend. If the data cannot be read, this method blocks the I/O read.	SQLException
readPending()	ByteBuffer	Reads the next WAL record from the backend. If the data cannot be read, this method does not block the I/O read.	SQLException
setAppliedLSN(LogSequenceNumber applied)	void	Sets the applied LSN.	-

Method Name	Return Type	Description	throws
setFlushedLSN(LogSequenceNumber flushed)	void	Updates the LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots), which is sent to the backend at the next update to update the LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots) on the server.	-

5.4.4.17 ChainedStreamBuilder

ChainedStreamBuilder is an API class provided by the GaussDB JDBC driver. It is used to build replication streams.

Inheritance Relationship of ChainedStreamBuilder

ChainedStreamBuilder is a logical replication API. Its implementation class is ReplicationStreamBuilder, which is in the **com.huawei.opengauss.jdbc.replication.fluent** package. The declaration of the class is as follows:

```
public class ReplicationStreamBuilder implements ChainedStreamBuilder
```

Constructor Method

```
public ReplicationStreamBuilder(final BaseConnection connection)
```

Common Methods

Table 5-30 Common methods of ReplicationStreamBuilder

Method Name	Return Type	Description	throws
logical()	ChainedLogicalStreamBuilder	Creates a logical replication stream.	-
physical()	ChainedPhysicalStreamBuilder	Creates a physical replication stream.	-

5.4.4.18 ChainedCommonStreamBuilder

ChainedCommonStreamBuilder is an API class provided by the GaussDB JDBC driver. It is used to specify common parameters for logical and physical replication.

Inheritance Relationship of ChainedCommonStreamBuilder

ChainedCommonStreamBuilder is an API for logical replication. The implementation abstract class is AbstractCreateSlotBuilder. The inheritance class is LogicalCreateSlotBuilder which is in the **com.huawei.opengauss.jdbc.replication.fluent.logical** package. The declaration of this class is as follows:

```
public class LogicalCreateSlotBuilder
    extends AbstractCreateSlotBuilder<ChainedLogicalCreateSlotBuilder>
    implements ChainedLogicalCreateSlotBuilder
```

Constructor Method

```
public LogicalCreateSlotBuilder(BaseConnection connection)
```

Common Methods

Table 5-31 Common methods of LogicalCreateSlotBuilder

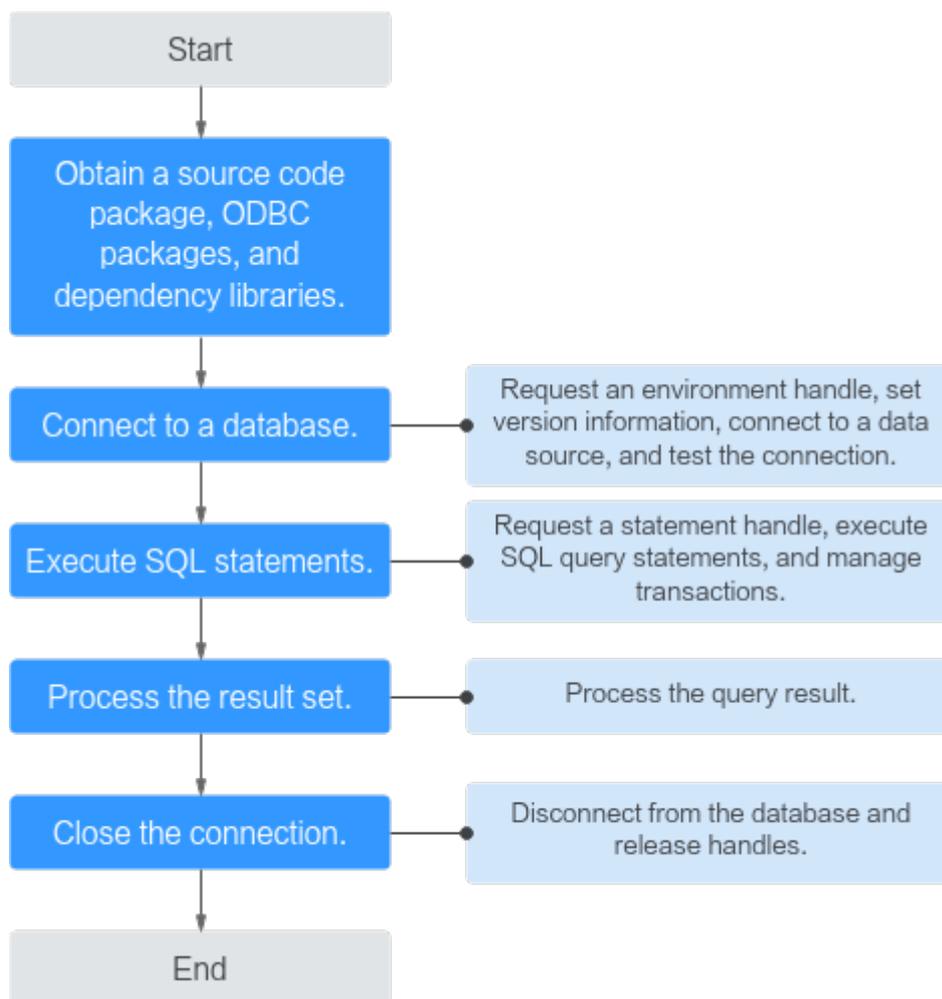
Method Name	Return Type	Description	throws
withSlotName(String slotName)	T	Specifies the name of a replication slot.	-
withOutputPlugin(String outputPlugin)	ChainedLogicalCreateSlotBuilder	Plug-in name. Currently, mppdb_decoding is supported. mppdb_decoding : a decoding output format. After this value is set, the output content is in JSON format. The output result contains the attribute information and attribute values of the related data.	-
make()	void	Creates a slot with the specified parameters in the database.	SQLException
self()	ChainedLogicalCreateSlotBuilder	Returns the implementation of ChainedLogicalCreateSlotBuilder.	-

5.5 Development Based on ODBC

5.5.1 Development Process

Figure 5-3 shows the ODBC-based development process.

Figure 5-3 ODBC development process



5.5.2 Development Process

5.5.2.1 Obtaining a Source Code Package, ODBC Packages, and Dependent Libraries

You need to obtain the unixODBC source code package, ODBC packages, and dependency libraries for ODBC-based development.

Obtaining the Source Code Package

- To obtain the unixODBC source code package, visit <https://www.unixodbc.org/unixODBC-2.3.7.tar.gz>.
- To obtain the MD5 file, visit <https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5>.

Use the MD5 file to verify the integrity of the unixODBC source code package.
(Check whether the MD5 value is the same as that in the source code package.)

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-32](#).

Table 5-32 Driver package download list

Version	Download Address
V2.0-3.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

- Verifying the software package integrity on a Linux server:
 - a. Upload the software package and verification package to the same directory on the VM.
 - b. Run the following command to verify the integrity of the software package:

```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

If **OK** is displayed in the command output, the verification is successful.

```
GaussDB_driver.zip: OK
```
- Verifying the software package integrity on a Windows server:
 - a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
 - b. Run the following command to obtain the hash value of the driver package:

```
certutil -hashfile {local_directory_of_the_driver_package}\  
{driver_package_name} sha256
```

 - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
 - Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **GaussDB_driver.zip**.Example: **certutil -hashfile C:\Users\GaussDB_driver.zip sha256**
 - c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in [Table 5-32](#).
 - If they are consistent, the verification is successful.
 - If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Obtaining ODBC Packages and Dependency Libraries

ODBC packages and dependency libraries support both Linux and Windows systems. Obtain the required files based on the OS you use.

- For Linux:

Obtain the package **GaussDB-Kernel_Database version number_OS information_64bit_Odbc.tar.gz** from the driver package. In the Linux OS, unixODBC header files (including **sql.h** and **sqlext.h**) and the library **libodbc.so** are required in application development.

The header files and the library can be obtained from the unixODBC-2.3.7 source code package.

- For Windows:

Obtain the package **GaussDB-Kernel_Database version number_Windows_X86_Odbc.tar.gz** (32-bit) or **GaussDB-Kernel_Database version number_Windows_X64_Odbc.tar.gz** (64-bit) from the driver package.

In the Windows OS, the required header files and library files are system-resident.

5.5.2.2 Connecting to a Database

5.5.2.2.1 Configuring a Data Source in the Linux OS

Before using ODBC to connect to a database, you need to prepare the required resources. To connect to a database, you need to configure the ODBC data source or use the ODBC APIs or the driver to implement the communication and interaction between an application and a database. This section describes how to configure a data source and connect to a database in the Linux OS.

Procedure

- Step 1** Install unixODBC. (By default, the unixODBC source code package has been obtained during environment preparation.) It does not matter if unixODBC of another version has been installed.

For example, install unixODBC-2.3.7.

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7
```

```
./configure --enable-gui=no # To perform compilation on an Arm server, add the configure parameter --  
build=aarch64-unknown-linux-gnu.  
make  
# The installation may require root permissions.  
make install
```

NOTE

- Currently, unixODBC-2.2.1 is not supported.
- By default, it is installed in the **/usr/local** directory. The data source file is generated in the **/usr/local/etc** directory, and the library file is generated in the **/usr/local/lib** directory.
- You can compile unixODBC with the **--enable-fastvalidate=yes** option to achieve higher performance. However, this option may cause an application that passes an invalid handle to the ODBC API to fail instead of returning an SQL_INVALID_HANDLE error.

Step 2 Replace the GaussDB client driver.

Decompress **GaussDB-Kernel_Database version number_OS information_64bit_Odbc.tar.gz**. After the decompression, the **lib** and **odbc** folders are generated. The **odbc** folder contains another **lib** folder. Copy all dynamic libraries in the **/lib** and **/odbc/lib** folders to the **/usr/local/lib** directory.

Step 3 Configure a data source.

1. Configure the ODBC driver file.

Add the following content to the **/usr/local/etc/odbcinst.ini** file:

```
[GaussMPP]
Driver64=/usr/local/lib/psqlodbcw.so
setup=/usr/local/lib/psqlodbcw.so
```

For descriptions of the parameters in the **odbcinst.ini** file, see [Table 5-33](#).

Table 5-33 odbcinst.ini configuration parameters

Parameter	Description	Example
[DriverName]	Driver name, corresponding to the driver in DSN.	[GaussMPP]
Driver64	Path of the dynamic driver library.	Driver64=/usr/local/lib/psqlodbcw.so
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/usr/local/lib/psqlodbcw.so

2. Configure the data source file.

Add the following content to the **/usr/local/etc/odbc.ini** file:

```
[gaussdb]
Driver=GaussMPP
Servername=127.0.0.1 (database server IP address)
Database=postgres (database name)
Username=omm (database username)
Password= (database user password)
Port=8000 (database listening port)
Sslmode=allow
```

For descriptions of the parameters in the **odbc.ini** file, see [Table 5-34](#).

Table 5-34 odbc.ini configuration parameters

Parameter	Description	Example
[DSN]	Data source name.	[gaussdb]
Driver	Driver name, corresponding to DriverName in odbcinst.ini .	Driver=GaussMPP

Parameter	Description	Example
Servername	Server IP address. Multiple IP addresses can be configured.	Servername=127.0.0.1
Database	Name of the database to connect to.	Database = postgres
Username	Database username.	Username=omm
Password	<p>Password of the database user.</p> <p>NOTE After a user establishes a connection, the ODBC driver automatically clears their password stored in memory.</p> <p>However, if this parameter is configured, unixODBC will cache data source files, which may cause the password to be stored in the memory for a long time.</p> <p>When you connect to an application, you are advised to send your password through an API instead of writing it in a data source configuration file. After the connection has been established, immediately clear the memory segment where your password is stored.</p> <p>NOTICE The password in the configuration file must comply with the following HTTP rules:</p> <ol style="list-style-type: none"> 1. Characters must comply with the URL encoding specifications. For example, the exclamation mark (!) must be written as %21, and the percent sign (%) must be written as %25. Therefore, pay attention to the characters. 2. A plus sign (+) will be replaced by a space. 	Password=*****

Parameter	Description	Example
Port	Port number of the server. When load balancing is enabled, multiple port numbers can be configured and must correspond to multiple IP addresses. If multiple IP addresses are configured and only one port number is configured when load balancing is enabled, all IP addresses share the same port number by default, that is, the configured port number.	Port=8000
Sslmode	Specifies whether to enable the SSL connection. NOTE For details about the values of the Sslmode option, see Table 5-35 .	Sslmode = allow
Debug	Specifies whether to enable the debugging mode. Value range: 0 to <i>INT_MAX</i> . - If this parameter is set to 0 , the function is disabled. - If this parameter is set to a value greater than 0, the mylog file of the psqLODBC driver will be printed. The directory generated for storing logs is /tmp/ . The default value is 0 .	Debug = 1

Parameter	Description	Example
UseServerSidePrepare	<p>Specifies whether to enable the extended query protocol for the database.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If this parameter is set to 0, the function is disabled. - If this parameter is set to 1, the function is enabled. <p>The default value is 1.</p>	UseServerSidePrepare = 1
UseBatchProtocol	<p>Specifies whether to enable the batch query protocol. If it is enabled, DML performance can be improved.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If this parameter is set to 0, the batch query protocol is disabled (mainly for communication with earlier database versions). - If this parameter is set to 1 and support_batch_bind is set to on, the batch query protocol is enabled. <p>The default value is 1.</p>	UseBatchProtocol = 1
ForExtensionConnector	<p>Specifies whether the savepoint is sent.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If this parameter is set to 0, the savepoint is sent. - If this parameter is set to 1, the savepoint is not sent. <p>The default value is 1.</p>	ForExtensionConnector = 1

Parameter	Description	Example
ConnectionExtraInfo	<p>Specifies whether to display the driver deployment path and process owner in the GUC parameter connection_info.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If this parameter is set to 0, the information is not displayed. - If this parameter is set to 1, the information is displayed. <p>The default value is 0.</p> <p>NOTE If this parameter is set to 1, the ODBC driver reports the driver deployment path and process owner to the database and displays the information in the GUC parameter connection_info. In this case, you can query the information from PG_STAT_ACTIVITY or PGXC_STAT_ACTIVITY.</p>	ConnectionExtraInfo = 1
BoolsAsChar	<p>Specifies whether to process Boolean values as characters.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If this parameter is set to 0, the Boolean values are mapped to the SQL_BIT type. - If this parameter is set to 1, the Boolean values are mapped to the SQL_CHAR type. <p>The default value is 1.</p>	BoolsAsChar = 1

Parameter	Description	Example
RowVersioning	<p>Specifies whether to allow the application to check whether a row of data is modified by another user when the row of data is updated.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If this parameter is set to 0, application detection is not allowed. - If this parameter is set to 1, application detection is allowed. <p>The default value is 0.</p>	RowVersioning=1
ShowSystemTables	<p>Specifies whether to regard the system catalog as a common SQL table by default.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If this parameter is set to 0, the driver does not regard the system catalog as a common SQL table by default. - If this parameter is set to 1, the driver regards the system catalog as a common SQL table by default. <p>The default value is 0.</p>	ShowSystemTables=1
AutoBalance	<p>Specifies whether to enable ODBC load balancing.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If the value is 0, load balancing is disabled. - If the value is 1, load balancing is enabled. <p>The default value is 0.</p>	AutoBalance=1

Parameter	Description	Example
RefreshCNListTime	<p>Time for refreshing the CN list. This parameter can be configured when load balancing is enabled. The value is an integer, in seconds.</p> <p>Value range: 0 to <i>INT_MAX</i>.</p> <ul style="list-style-type: none"> - If the value is 0, this parameter is disabled. - If the value is greater than 0, the value is the time for refreshing the CN list. <p>The default value is 10.</p>	RefreshCNListTime=5
Priority	<p>This parameter can be configured when load balancing is enabled.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If the value is 0, this parameter is disabled. - If the value is 1, this parameter is enabled. <p>The default value is 0.</p> <p>NOTE When Priority is enabled, all connections initiated by applications are preferentially sent to the CNs configured in the configuration file. If all the configured CNs are unavailable, the connections are sent to the remaining CNs.</p>	Priority=1

Parameter	Description	Example
UsingEip	<p>This parameter can be configured when load balancing is enabled.</p> <p>Value range: 0 and 1</p> <ul style="list-style-type: none"> - If the value is 0, this parameter is disabled. - If the value is 1, it is enabled. <p>The default value is 0.</p> <p>NOTE This value specifies whether to use an elastic IP address for load balancing. If UsingEip is enabled, the elastic IP address is used for load balancing. If UsingEip is disabled, the data IP address is used for load balancing.</p>	UsingEip=1
MaxCacheQueries	<p>Controls the number of prepared statements cached for each connection.</p> <p>Value range: 0 to 4096</p> <p>The default value is 0.</p> <p>NOTE If this parameter is set to 0, the prepared statement cache pool is disabled on the client. If this parameter is set to a value greater than 4096, the value 4096 is used. If the number of executed statements exceeds the upper limit specified by MaxCacheQueries, the least recently used statements are eliminated.</p>	MaxCacheQueries=128

Parameter	Description	Example
MaxCacheSizeMiB	<p>Controls the total size of prepared statements cached for each connection. This parameter takes effect when the value of MaxCacheQueries is greater than 0.</p> <p>Value range: 0 to 4096</p> <p>The default value is 1.</p> <p>NOTE If the total size of cached statements is greater than the value of MaxCacheSizeMiB, the least recently used statements are eliminated. If this parameter is set to a value greater than 4096, the value 4096 is used. The unit is MB.</p>	MaxCacheSizeMiB=10
TcpUserTimeout	<p>Specifies the maximum duration for which transmitted data can remain unacknowledged before the TCP connection is forcibly closed on an OS that supports the TCP_USER_TIMEOUT socket option.</p> <p>Value range: 0 to <i>INT_MAX</i>.</p> <p>The default value is 0.</p> <p>NOTE 0 indicates that the default value is used. Ignore this parameter for UDS connections. The unit is millisecond.</p>	TcpUserTimeout=5000

Table 5-35 Sslmode options

Sslmode	Enable SSL Encryption	Description
disable	No	SSL connection is not enabled.

Sslmode	Enable SSL Encryption	Description
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by verify-ca , the system checks whether the name of the host where the database resides is the same as that in the certificate. GaussDB does not support this mode.

 **NOTE**

When establishing connections to the GaussDB server using ODBC, you can enable SSL connections to encrypt client and server communications. To enable SSL connection, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

Step 4 Configure the environment variables on the client.

```
vim ~/.bashrc
```

Add the following information to the configuration file:

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

Step 5 Run the following command to validate the addition:

```
source ~/.bashrc
```

Step 6 Test the connection.

After the installation, the generated binary file is stored in the **/usr/bin** directory. You can run the **isql -v gaussdb** command (*gaussdb* is the data source name).

- If the following information is displayed, the configuration is correct and the connection succeeds:

```
+-----+
| Connected!          |
|                    |
| sql-statement      |
| help [tablename]   |
| quit              |
```

- If error information is displayed, the configuration is incorrect. Check the configuration.
- In a cluster environment, you need to copy and configure the unixODBC file on all nodes.

 **NOTE**

When ODBC is used to connect to a database, the kernel parameters are set as follows:
SET extra_float_digits = 2;
SET DateStyle = 'ISO';

These parameters may cause the ODBC and gsql clients to display inconsistent data in aspects such as date data display mode and floating-point precision representation. If the result is not as expected, you are advised to explicitly set these parameters in the ODBC application code.

----End

Troubleshooting

- [UnixODBC][Driver Manager]Can't open lib 'xxx/xxx/psqlodbcw.so':file not found.
Possible causes:
 - The path configured in the **odbcinst.ini** file is incorrect.
Run **ls** to check the path in the error information, ensuring that the **psqlodbcw.so** file exists and you have execution permissions on it.
 - The dependent library of **psqlodbcw.so** does not exist or is not in system environment variables.
Run the **ldd** command to check the path in the error information. If the unixODBC library such as **libodbc.so.1** is missing, reconfigure unixODBC according to the procedure, ensure that the **lib** directory in the installation path is added to **LD_LIBRARY_PATH**. If the problem persists after reinstallation, manually copy the content in the **unixodbc/lib** directory of the database installation package to the **lib** directory in the installation path of the unixODBC. If other libraries do not exist, add the **lib** directory under the ODBC driver package to **LD_LIBRARY_PATH**. If other standard libraries are missing, install them.
- [UnixODBC]connect to server failed: no such file or directory
Possible causes:
 - An incorrect or unreachable database IP address or port was configured.
Check the **Servername** and **Port** configuration items in data sources.
 - Server monitoring is improper.
If **Servername** and **Port** are correctly configured, ensure the proper NIC and port are monitored by following the database server configurations in the procedure in this section.
 - Firewall and network gatekeeper settings are improper.
Check firewall settings, and ensure that the database communication port is trusted.
Check to ensure that network gatekeeper settings are proper (if any).

- [unixODBC]The password-stored method is not supported.
Possible causes:
The **Sslmode** configuration item is not configured in the data sources.
Solution:
Set the configuration item to **allow** or a higher level. For details, see [Table 5-35](#).
- Server common name "xxxx" does not match host name "xxxxx"
Possible causes:
When **verify-full** is used for SSL encryption, the driver checks whether the host name in certificates is the same as the actual one.
Solution:
To solve this problem, use **verify-ca** to stop checking host names, or generate a set of server certificates containing the actual host names.
- Driver's SQLAllocHandle on SQL_HANDLE_DBC failed
Possible causes:
The executable file (such as the **isql** tool of unixODBC) and the database driver (**psqlodbcw.so**) depend on different library versions of ODBC, such as **libodbc.so.1** and **libodbc.so.2**. You can verify this problem by using the following method:

```
ldd `which isql` | grep odbc
ldd psqlodbcw.so | grep odbc
```

If the suffix digits of the outputs **libodbc.so** are different or indicate different physical disk files, this problem exists. Both **isql** and **psqlodbcw.so** require **libodbc.so** to be loaded. If they load different physical files, two sets of function lists with the same name are generated in a visible domain (the **libodbc.so.*** function export lists of unixODBC are the same). This results in conflicts and the database driver cannot be loaded.
Solution:
Uninstall the unnecessary unixODBC, such as **libodbc.so.2**, and create a soft link with the same name and the **.so.2** suffix for the remaining **libodbc.so.1** library.
- FATAL: Forbid remote connection with trust method!
For security reasons, the database CN prevents unauthorized access from other nodes within the cluster.
To access the CN from inside the cluster, deploy the ODBC program on the host where the CN is located and use 127.0.0.1 as the server address. It is recommended that the service system be deployed outside the cluster. If it is deployed inside, database performance may be affected.
- [unixODBC][Driver Manager]Invalid attribute value
This problem occurs when you use SQL on other GaussDB. The possible cause is that the unixODBC version is not the recommended one. You are advised to run the **odbcinst --version** command to check the unixODBC version.
- authentication method 10 not supported.
If this error occurs on an open-source client, the cause may be:
The database stores only the SHA-256 hash of the password, but the open-source client supports only MD5 hashes.

 NOTE

- The database stores the hashes of user passwords instead of actual passwords.
- If a password is updated or a user is created, both types of hashes will be stored, compatible with open-source authentication protocols.
- An MD5 hash can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash. Passwords in the source version will only have SHA-256 hashes and not support MD5 authentication.
- MD5 has lower security and poses security risks. Therefore, you are advised to use a more secure cryptographic algorithm.

To solve this problem, you can update the user password (see [ALTER USER](#)) or create a user (see [CREATE USER](#)) having the same permissions as the faulty user.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0
The database version is too early or the database is an open-source database. Use the driver of the required version to connect to the database.
- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.
In `pg_hba.conf` of the target CN, the authentication mode is set to `gss` for authenticating the IP address of the current client. However, this authentication algorithm cannot authenticate clients. Change the authentication algorithm to `sha256` and try again.
- isql: error while loading shared libraries:xxx
The dynamic library does not exist in the environment. You need to install the corresponding library.

5.5.2.2 Configuring a Data Source in the Windows OS

Procedure

Step 1 Replace the GaussDB client driver.

Decompress the `GaussDB-Kernel_Database version number_Windows_X64_Odbc.tar.gz` (64-bit) driver package or `GaussDB-Kernel_Database version number_Windows_X86_Odbc.tar.gz` (32-bit) driver package, and click `psqlodbc.exe` to install the driver.

Step 2 Open the driver manager.

Use the ODBC driver manager corresponding to the ODBC version. If the 64-bit ODBC driver is used, the 64-bit ODBC driver manager must be used. Assume that the OS is installed on drive C (if the OS is installed on another drive, change the path accordingly):

- If you want to use 32-bit ODBC driver manager in a 64-bit OS, open `C:\Windows\SysWOW64\odbcad32.exe`. Do not choose **Control Panel > Administrative Tools > Data Sources (ODBC)**.

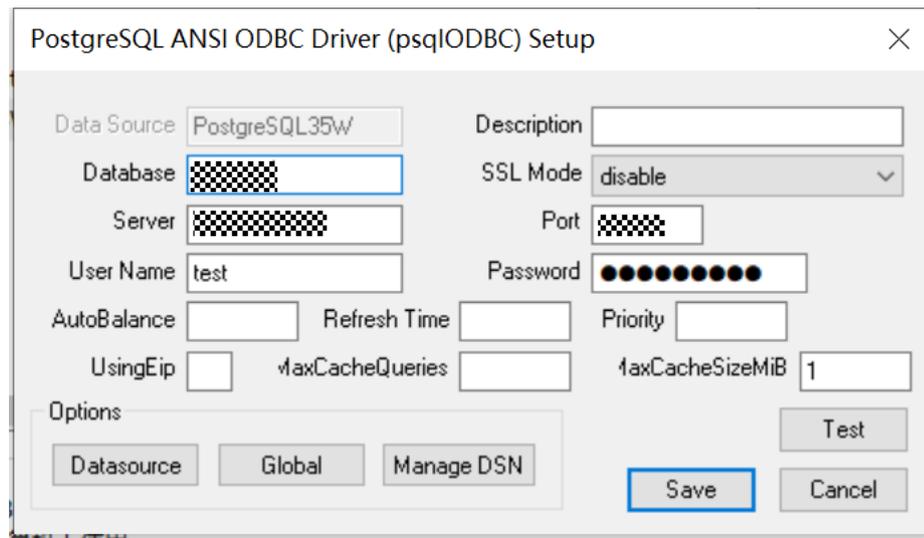
 NOTE

WoW64 is short for Windows 32-bit on Windows 64-bit. `C:\Windows\SysWOW64\` stores the 32-bit environment on a 64-bit system. `C:\Windows\System32\` stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- For a 32-bit OS, open **C:\Windows\System32\odbcad32.exe** or choose **Computer > Control Panel > Administrative Tools > Data Sources (ODBC)** to open Driver Manager.
- For a 64-bit OS, choose **Control Panel > Administrative Tools > Data Sources (ODBC)** to enable driver management.

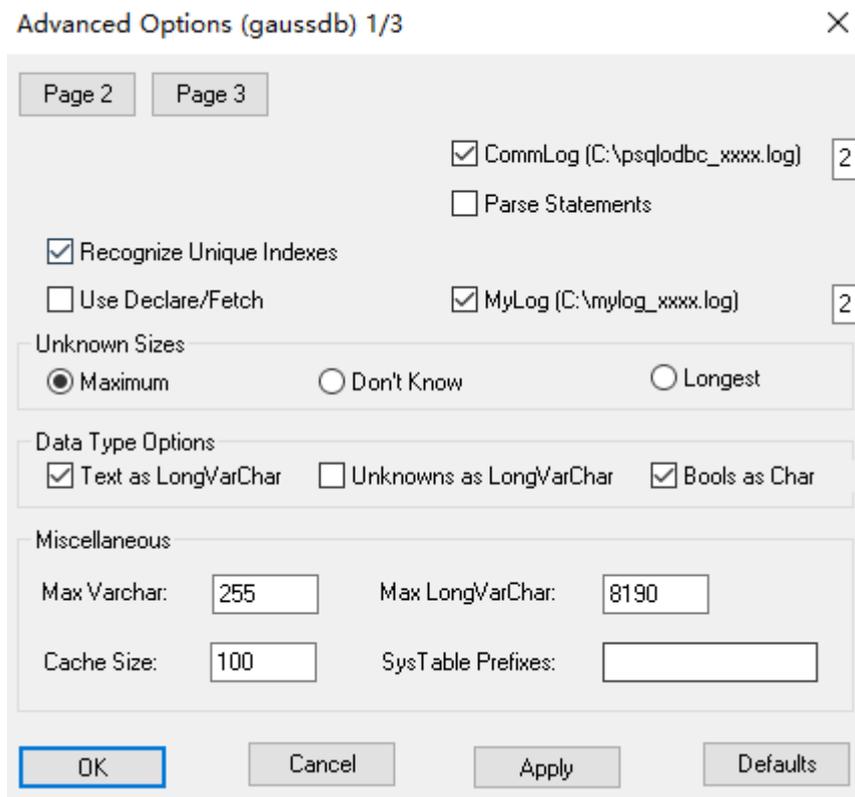
Step 3 Configure a data source.

On the **User DSN** tab, click **Add** and choose **PostgreSQL Unicode** for setup.



For details about the parameters, see [Configuring a Data Source in the Linux OS](#).

You can click **Datasource** to configure whether to print logs.



NOTICE

The entered username and password will be recorded in the Windows registry and you do not need to enter them again when connecting to the database next time. For security purposes, you are advised to delete sensitive information before clicking **Save** and enter the required username and password again when using ODBC APIs to connect to the database.

Step 4 Enable the SSL mode.

Change the value of **SSL Mode** in **Step 3** to **require**.

Table 5-36 Sslmode options

Sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.

SSLmode	Whether SSL Encryption Is Enabled	Description
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified. Currently, Windows ODBC does not support the certificate-based authentication.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by verify-ca , the system checks whether the name of the host where the database resides is the same as that in the certificate. Currently, Windows ODBC does not support the certificate-based authentication.

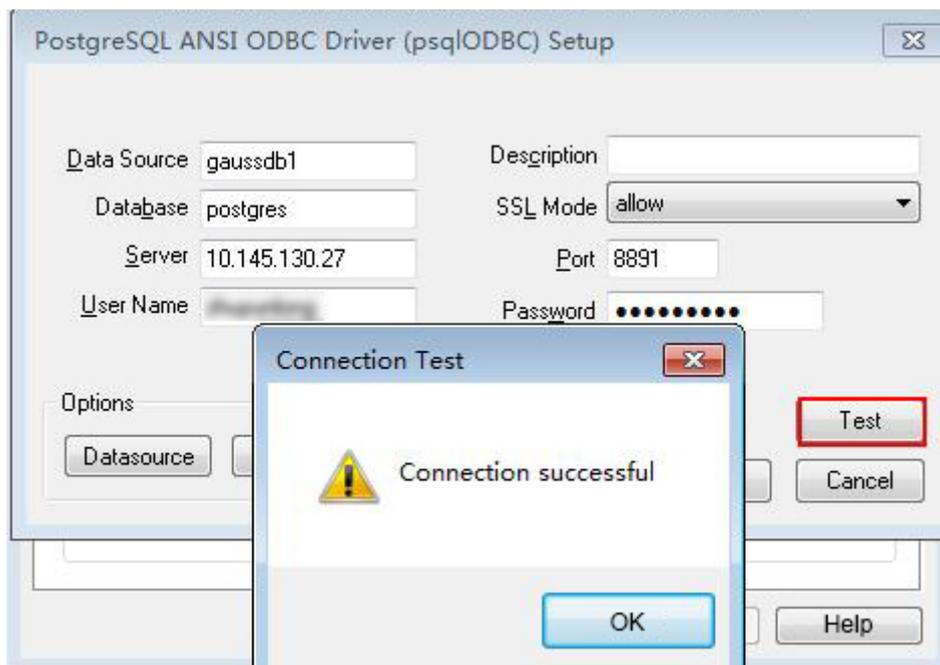
 **NOTE**

When establishing connections to the GaussDB server using ODBC, you can enable SSL connections to encrypt client and server communications. To enable SSL connection, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

Step 5 Test the connection.

Click **Test**.

- If the following information is displayed, the configuration is correct and the connection succeeds.



- If error information is displayed, the configuration is incorrect. Check whether the preceding configurations are correct.

NOTE

When ODBC is used to connect to a database, the kernel parameters are set as follows:
`SET extra_float_digits = 2;`
`SET DateStyle = 'ISO';`

These parameters may cause the ODBC and gsql clients to display inconsistent data in aspects such as date data display mode and floating-point precision representation. If the result is not as expected, you are advised to explicitly set these parameters in the ODBC application code.

----End

Troubleshooting

- connect to server failed: no such file or directory
 Possible causes:
 - An incorrect or unreachable database IP address or port was configured.
 Check the **Server** and **Port** configuration items in data sources.
 - Server monitoring is improper.
 If **Server** and **Port** are correctly configured, ensure the proper NIC and port are monitored by following the database server configurations in the procedure in this section.
 - Firewall and network gatekeeper settings are improper.
 Check firewall settings, and ensure that the database communication port is trusted.
 Check to ensure that network gatekeeper settings are proper (if any).
- The password-stored method is not supported.
 Possible causes:

Sslmode is not configured for the data source. Set this configuration item to **allow** or a higher level to enable SSL connections. For details on **Sslmode**, see [Table 5-36](#).

- authentication method 10 not supported.

If this error occurs on an open-source client, the cause may be:

The database stores only the SHA-256 hash of the password, but the open-source client supports only MD5 hashes.

 **NOTE**

- The database stores the hashes of user passwords instead of actual passwords.
- If a password is updated or a user is created, both types of hashes will be stored, compatible with open-source authentication protocols.
- An MD5 hash can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash. Passwords in the source version will only have SHA-256 hashes and not support MD5 authentication.
- The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.

To solve this problem, you can update the user password (see [ALTER USER](#)) or create a user (see [CREATE USER](#)) having the same permissions as the faulty user.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

The database version is too early or the database is an open-source database. Use the driver of the required version to connect to the database.

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

In **pg_hba.conf** of the target CN, the authentication mode is set to **gss** for authenticating the IP address of the current client. However, this authentication algorithm cannot authenticate clients. Change the authentication algorithm to **sha256** and try again.

5.5.2.2.3 APIs for Connecting to a Database

After the database connection test is successful, the ODBC APIs provide a group of functions to connect to the database, as shown in [Table 5-37](#).

Table 5-37 API description

Function	API
Allocate a handle.	SQLAllocHandle is a generic function for allocating a handle. It can replace the following functions: <ul style="list-style-type: none"> • SQLAllocEnv: allocates an environment handle. • SQLAllocConnect: allocates a connection handle. • SQLAllocStmt: allocates a statement handle.
Set environment attributes.	SQLSetEnvAttr
Set connection attributes.	SQLSetConnectAttr

Function	API
Connect to a database.	SQLConnect

The following uses the development source program **DBtest.c** as an example (for details about the complete example, see [Obtaining and Processing Data in a Database](#)).

```
// DBtest.c (compile with: libodbc.so)
// For details about program header files and global variables, see the complete example.

// Allocate an environment handle.
V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
    printf("Error AllocHandle\n");
    exit(0);
}

// Set the version information (environment attributes).
SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

// Allocate a connection handle.
V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    exit(0);
}

// Obtain the username and password.
char *userName;
userName = getenv("EXAMPLE_USERNAME_ENV");
char *password;
password = getenv("EXAMPLE_PASSWORD_ENV");

// Set connection attributes.
SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,(SQLPOINTER *)SQL_AUTOCOMMIT_ON, 0);

// Connect to the database. userName and password indicate the username and password for connecting
to the database, respectively.
// If the username and password have been set in the odbc.ini file, you can retain "". However, you are
advised not to do so because the username and password will be disclosed if the permission for odbc.ini is
abused.
V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
    (SQLCHAR*) userName, SQL_NTS, (SQLCHAR*) password, SQL_NTS);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
    printf("Error SQLConnect %d\n",V_OD_erg);
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    exit(0);
}
printf("Connected !\n");
```

5.5.2.3 Running SQL Statements

To help users interact with the database, the ODBC provides APIs for executing SQL statements, as shown in [Table 5-38](#).

Table 5-38 API description

Function	API
Set statement attributes.	SQLSetStmtAttr
Prepare an SQL statement for execution.	SQLPrepare
Run a prepared SQL statement.	SQLExecute
Bind the parameter marker of an SQL statement to a buffer.	SQLBindParameter
Run an SQL statement directly.	SQLExecDirect

NOTE

- ODBC connects applications to the database and delivers the SQL statements sent by an application to the database. It does not parse the SQL syntax. Therefore, when confidential information (such as a plaintext password) is written into the SQL statement sent by an application, the confidential information is exposed in the driver log.
- If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. If one of the statements fails, the entire request will be rolled back.

The following is an example (for details about the complete example, see [Obtaining and Processing Data in a Database](#)):

```
// Set statement attributes.
SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT, (SQLPOINTER *)3, 0);

// Allocate a statement handle.
SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);

// Run an SQL statement directly.
SQLExecDirect(V_OD_hstmt, "drop table IF EXISTS customer_t1", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "insert into customer_t1 values(25, 'li')", SQL_NTS);

// Prepare for execution.
SQLPrepare(V_OD_hstmt, "insert into customer_t1 values(?)", SQL_NTS);

// Add parameters.
SQLBindParameter(V_OD_hstmt,1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0,
&value, 0, NULL);

// Run the prepared statement.
SQLExecute(V_OD_hstmt); SQLExecDirect(V_OD_hstmt, "select c_customer_sk from customer_t1",
SQL_NTS);
```

5.5.2.4 Processing Data in a Result Set

The ODBC obtains data from the database and provides the data to the application program for processing. Functions of ODBC processing include but are

not limited to: retrieving data, displaying data, processing data, transmitting data, and implementing service logic.

The ODBC provides APIs for processing data in a result set, as described in [Table 5-39](#).

Table 5-39 API description

Function	API
Bind a buffer to a column in the result set.	SQLBindCol
Fetch the next row (or rows) from a result set.	SQLFetch
Return data in a column of a result set.	SQLGetData
Get the column information from a result set.	SQLColAttribute
Return the error message of the last operation.	SQLGetDiagRec

The following is an example (for details about the complete example, see [Obtaining and Processing Data in a Database](#)):

```
// After the SQL statement is executed, obtain the attributes of a column in the result set.
SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
printf("SQLColAttribute %s\n",typename);

// Bind the result set.
SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
          (SQLLEN *)&V_OD_err);

// Use SQLFetch to obtain data from the result set.
V_OD_erg=SQLFetch(V_OD_hstmt);

// Use SQLGetData to obtain and return data.
while(V_OD_erg != SQL_NO_DATA)
{
    SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
    printf("SQLGetData ----ID = %d\n",V_OD_id);
    V_OD_erg=SQLFetch(V_OD_hstmt);
};
printf("Done !\n");
```

5.5.2.5 Closing a Connection

The ODBC provides APIs for disconnecting from a database and releasing resources, as described in [Table 5-40](#).

Table 5-40 API description

Function	API
Disconnect from a data source.	SQLDisconnect
Release a handle.	<p>SQLFreeHandle is a generic function for releasing a handle. It can replace the following functions:</p> <ul style="list-style-type: none"> • SQLFreeEnv: releases an environment handle. • SQLFreeConnect: releases a connection handle. • SQLFreeStmt: releases a statement handle.

The following is an example (for details about the complete example, see [Obtaining and Processing Data in a Database](#)):

```
// Disconnect from the data source and release handles.
SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
SQLDisconnect(V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
```

5.5.3 Typical Application Development Examples

5.5.3.1 Typical Application Scenarios and Configurations

Log Diagnosis

ODBC logs are classified into unixODBC driver manager logs and psqLODBC driver logs. The former is used to trace whether the application API is successfully executed, and the latter is used to locate problems based on DFX logs generated during underlying implementation.

The unixODBC log needs to be configured in the **odbcinst.ini** file:

```
[ODBC]
Trace=Yes
TraceFile=/path/to/odbctrace.log

[GaussMPP]
Driver64=/usr/local/lib/psqlodbcw.so
setup=/usr/local/lib/psqlodbcw.so
```

For psqLODBC logs, you only need to add the following content to **odbc.ini**:

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13 # Database server IP address
...
Debug=1 # Enable the debug log function of the driver.
```

NOTE

The unixODBC logs are generated in the path configured by **TraceFile**. The psqLODBC generates the **mylog_XXX.log** file in the **/tmp/** directory.

Load Balancing

Load balancing can be enabled when there are a large number of concurrent applications.

- Load balancing is to randomly distribute concurrent connections to all CNs to prevent a single CN from being overloaded and improve performance.
- Set **AutoBalance** to **1** to enable load balancing.
- Set **RefreshCNListTime** to **5** as required. The default refresh interval is 10s.
- Set **Priority** to **1** as required. In this case, concurrent connections are preferentially sent to the CNs configured in the configuration file. If all the configured CNs are unavailable, the connections are distributed to the remaining CNs.

Example:

Six CNs, namely, CN1, CN2, CN3, CN4, CN5, and CN6, are configured in the cluster, and four CNs, namely, CN1, CN2, CN3, and CN4, are configured in the configuration file.

Example content of the **odbc.ini** file:

```
[gaussdb]
Driver=GaussMPP
Servername=10.145.130.26,10.145.130.27,10.145.130.28,10.145.130.29 # IP address of the database server.
Database=postgres # Database name.
Username=omm # Database username.
Password= # Database user password.
Port=8000 # Database listening port.
Sslmode=allow
AutoBalance=1
RefreshCNListTime=3
Priority=1
```

If the configuration file and cluster environment are the same as those in the example, concurrent connections are randomly and evenly distributed to CN1, CN2, CN3, and CN4. When CN1, CN2, CN3, and CN4 are all unavailable, concurrent connections are randomly and evenly sent to CN5 and CN6. If any CN among CN1, CN2, CN3, and CN4 becomes available, the connections are not sent to CN5 and CN6 but to the available CN.

5.5.3.2 Obtaining and Processing Data in a Database

NOTE

- In Windows, ODBC application code can be compiled using the Minimalist GNU for Windows (MinGW) compiler. The compilation command is as follows:

```
gcc odbctest.c -o odbctest -lodb32
```

Run the following command:

```
./odbctest.exe
```

- In Linux, ODBC application code can be compiled using the GNU Compiler Collection (GCC). The compilation command is as follows:

```
gcc odbctest.c -o odbctest -lodb
```

Run the following command:

```
./odbctest
```

If **sql.h** or API cannot be found during compilation, manually connect to the header file and dynamic library of unixODBC.

```
gcc -I /home/omm/unixodbc/include -L /home/omm/unixodbc/lib odbctest.c -o odbctest -lodb
```

This example illustrates how to obtain and process data in GaussDB through ODBC.

Prerequisite: The data source has been configured successfully. For Linux OS, see [Configuring a Data Source in the Linux OS](#). For Windows OS, see [Configuring a Data Source in the Windows OS](#).

```
// DBtest.c (compile with: libodbc.so)
// In this example, the username and password are stored in environment variables. Before running this
// example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the
// local environment.
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
SQLHENV    V_OD_Env;    // Handle ODBC environment
SQLHSTMT   V_OD_hstmt; // Handle statement
SQLHDBC    V_OD_hdbc;  // Handle connection
char       typename[100];
SQLINTEGER value = 100;
SQLINTEGER V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
    // 1. Allocate an environment handle.
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
    }

    // 2. Set the version information (environment attributes).
    SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

    // 3. Allocate a connection handle.
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }

    // Obtain the username and password.
    char *userName;
    userName = getenv("EXAMPLE_USERNAME_ENV");
    char *password;
    password = getenv("EXAMPLE_PASSWORD_ENV");

    // 4. Set connection attributes.
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,(SQLPOINTER *)SQL_AUTOCOMMIT_ON, 0);

    // 5. Connect to the database. userName and password indicate the username and password for
    // connecting to the database, respectively.
    // If the username and password have been set in the odbc.ini file, you do not need to set userName or
    // password here, retaining "" for them. However, you are advised not to do so because the username and
    // password will be disclosed if the permission for odbc.ini is abused.
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
        (SQLCHAR*) userName, SQL_NTS, (SQLCHAR*) password, SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error SQLConnect %d\n",V_OD_erg);
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    printf("Connected !\n");
}
```

```
// 6. Set statement attributes.
SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT, (SQLPOINTER *)3,0);

// 7. Allocate a statement handle.
SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);

// 8. Run SQL statements.
SQLExecDirect(V_OD_hstmt, "drop table IF EXISTS customer_t1", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "insert into customer_t1 values(25,'li')", SQL_NTS);

// 9. Prepare for execution.
SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);

// 10. Bind parameters.
SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
&value,0,NULL);

// 11. Run prepared statements.
SQLExecute(V_OD_hstmt);
SQLExecDirect(V_OD_hstmt,"select c_customer_sk from customer_t1",SQL_NTS);

// 12. Obtain attributes of a specific column in the result set.
SQLColAttribute(V_OD_hstmt,1, SQL_DESC_TYPE,typename,100, NULL, NULL);
printf("SQLColAttribute %s\n",typename);

// 13. Bind the result set.
SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
(SQLLEN *)&V_OD_err);

// 14. Obtain data in the result set by executing SQLFetch.
V_OD_erg=SQLFetch(V_OD_hstmt);

// 15. Obtain and return data by executing SQLGetData.
while(V_OD_erg != SQL_NO_DATA)
{
    SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
    printf("SQLGetData ----ID = %d\n",V_OD_id);
    V_OD_erg=SQLFetch(V_OD_hstmt);
};
printf("Done !\n");

// 16. Disconnect data source connections and release handles.
SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
SQLDisconnect(V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
return(0);
}
```

The running result is as follows:

```
Connected !
SQLColAttribute
SQLGetData ----ID = 25
SQLGetData ----ID = 100
Done!
```

5.5.3.3 Batch Binding

Prerequisite: The data source has been configured successfully. For Linux OS, see [Configuring a Data Source in the Linux OS](#). For Windows OS, see [Configuring a Data Source in the Windows OS](#).

```
*****
* Enable UseBatchProtocol in the data source and set the database parameter support_batch_bind to on.
* The CHECK_ERROR command is used to check and print error information.
```

```
* This example is used to interactively obtain the DSN, volume of data to be processed, and volume of
ignored data from users, and insert required data into the test_odbc_batch_insert table.
*****/
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;          // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT; // Statement handle
    SQLCHAR loginfo[2048];

    // Allocate a statement handle.
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle(SQL_HANDLE_STMT) failed");
        return;
    }

    // Prepare a statement.
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS) failed");
        return;
    }

    // Execute the statement.
    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmt) failed");
        return;
    }

    // Release the handle.
    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLFreeHandle(SQL_HANDLE_STMT, hstmt) failed");
        return;
    }
}
int main ()
{
    SQLHENV henv = SQL_NULL_HENV;
    SQLHDBC hdbc = SQL_NULL_HDBC;
    long int batchCount = 1000; // Amount of data that is bound in batches
    SQLLEN rowsCount = 0;
    int ignoreCount = 0; // Amount of data that is not imported to the database among the data
that is bound in batches
    int i = 0;
    SQLRETURN retcode;
    SQLCHAR dsn[1024] = {'\0'};
    SQLCHAR loginfo[2048];
    do
    {
        if (ignoreCount > batchCount)
        {
            printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
        }
    }
}
```

```
    }
}while(ignoreCount > batchCount);

// Allocate an environment handle.
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}

// Set the ODBC version.
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER*)SQL_OV_ODBC3, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetEnvAttr failed");
    goto exit;
}

// Allocate connections.
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}

// Set login timeout.
retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetConnectAttr failed");
    goto exit;
}

// Set the automatic commit.
retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                            (SQLPOINTER)(1), 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetConnectAttr failed");
    goto exit;
}

// Connect to the database.
sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
retcode = SQLConnect(hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                    (SQLCHAR*) NULL, 0, NULL, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLConnect failed");
    goto exit;
}

// Initialize the table information.
Exec(hdbc, "drop table if exists test_odbc_batch_insert");
Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");
// The following code constructs the data to be inserted based on the data volume entered by users:
{
    SQLRETURN retcode;
    SQLHSTMT hstmtinsrt = SQL_NULL_HSTMT;
    SQLCHAR *sql = NULL;
    SQLINTEGER *ids = NULL;
    SQLCHAR *cols = NULL;
    SQLLEN *bufLenIds = NULL;
    SQLLEN *bufLenCols = NULL;
    SQLUSMALLINT *operptr = NULL;
    SQLUSMALLINT *statusptr = NULL;
```

```
SQLULEN    process = 0;

// Data is constructed by column. Each column is stored continuously.
ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);

// Data size in each row for a column
bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);

// Specifies whether this row needs to be processed. The value is SQL_PARAM_IGNORE or
SQL_PARAM_PROCEED.
operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
memset(operptr, 0, sizeof(operptr[0]) * batchCount);

// Processing result of the row
// Note: In the database, a statement belongs to one transaction. Therefore, data is processed as a
unit. Either all data is inserted successfully or all data fails to be inserted.
statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);
if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
{
    fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
    goto exit;
}
for (i = 0; i < batchCount; i++)
{
    ids[i] = i;
    sprintf(cols + 50 * i, "column test value %d", i);
    bufLenIds[i] = sizeof(ids[i]);
    bufLenCols[i] = strlen(cols + 50 * i);
    operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
}

// Allocate a statement handle.
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}

// Prepare a statement.
sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLPrepare failed");
    goto exit;
}
retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetStmtAttr failed");
    goto exit;
}
retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLBindParameter failed");
    goto exit;
}
retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50,
cols, 50, bufLenCols);

if (!SQL_SUCCEEDED(retcode)) {
```

```
        printf("SQLBindParameter failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLExecute(hstmtinesrt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmtinesrt) failed");
        goto exit;
        retcode = SQLRowCount(hstmtinesrt, &rowCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLRowCount failed");
            goto exit;
        }
        if (rowCount != (batchCount - ignoreCount))
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowCount(%d)", (batchCount -
ignoreCount), rowCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        else
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowCount(%d)", (batchCount -
ignoreCount), rowCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }

        // Check the number of returned rows.
        if (rowCount != process)
        {
            sprintf(loginfo, "process(%d) != rowCount(%d)", process, rowCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        else
        {
```

```
    sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute failed");
        goto exit;
    }
}
for (i = 0; i < batchCount; i++)
{
    if (i < ignoreCount)
    {
        if (statusptr[i] != SQL_PARAM_UNUSED)
        {
            sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
    }
    else if (statusptr[i] != SQL_PARAM_SUCCESS)
    {
        sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLExecute failed");
            goto exit;
        }
    }
}
retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLFreeHandle failed");
    goto exit;
}
}
}
exit:
(void) printf ("\nComplete.\n");

// Close the connection.
if (hdbc != SQL_NULL_HDBC) {
    SQLDisconnect(hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
}
// Release an environment handle.
if (henv != SQL_NULL_HENV)
    SQLFreeHandle(SQL_HANDLE_ENV, henv);
return 0;
}
```

The running result is as follows:

```
Complete.
```

Part of database query result is as follows:

id	col
0	column test value 0
1	column test value 1
2	column test value 2
3	column test value 3
4	column test value 4
5	column test value 5
6	column test value 6
7	column test value 7

```
8 | column test value 8
...
```

5.5.3.4 High-Performance Binding Type

If a large amount of data needs to be inserted, you are advised to perform the following operations:

- You need to set **UseBatchProtocol** to **1** in the **odbc.ini** file and **support_batch_bind** to **on** in the database.
- The ODBC program binding type must be the same as that in the database.
- The character set of the client is the same as that of the database.
- The transaction is committed manually.

odbc.ini configuration file:

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13 # Database server IP address
...
UseBatchProtocol=1 # Enabled by default
ConnSettings=set client_encoding=UTF8 # Set the character code on the client to be the same as that on
the server.
```

The binding type example is as follows:

Prerequisite: The data source has been configured successfully. For Linux OS, see [Configuring a Data Source in the Linux OS](#). For Windows OS, see [Configuring a Data Source in the Windows OS](#).

```
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
#include <sys/time.h>

#define MESSAGE_BUFFER_LEN 128
SQLHANDLE h_env = NULL;
SQLHANDLE h_conn = NULL;
SQLHANDLE h_stmt = NULL;
void print_error()
{
    SQLCHAR Sqlstate[SQL_SQLSTATE_SIZE+1];
    SQLINTEGER NativeError;
    SQLCHAR MessageText[MESSAGE_BUFFER_LEN];
    SQLSMALLINT TextLength;
    SQLRETURN ret = SQL_ERROR;

    ret = SQLGetDiagRec(SQL_HANDLE_STMT, h_stmt, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n STMT ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_DBC, h_conn, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n CONN ERROR-%05d %s", NativeError, MessageText);
```

```
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_ENV, h_env, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n ENV ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    return;
}

/* Expect the function to return SQL_SUCCESS. */
#define RETURN_IF_NOT_SUCCESS(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the function to return SQL_SUCCESS. */
#define RETURN_IF_NOT_SUCCESS_I(i, func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u (i=%d): : expect SQL_SUCCESS, but ret = %d", __LINE__, (i), ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the function to return SQL_SUCCESS_WITH_INFO. */
#define RETURN_IF_NOT_SUCCESS_INFO(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS_WITH_INFO != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS_WITH_INFO, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the values are the same. */
#define RETURN_IF_NOT(expect, value) \
if ((expect) != (value))\
{\
    printf("\n failed line = %u: expect = %u, but value = %u", __LINE__, (expect), (value)); \
    return SQL_ERROR;\
}

/* Expect the character strings are the same. */
#define RETURN_IF_NOT_STRCMP_I(i, expect, value) \
if (( NULL == (expect) ) || (NULL == (value)))\
{\
    printf("\n failed line = %u (i=%u): input NULL pointer !", __LINE__, (i)); \
    return SQL_ERROR; \
}\
else if (0 != strcmp((expect), (value)))\
{\
    printf("\n failed line = %u (i=%u): expect = %s, but value = %s", __LINE__, (i), (expect), (value)); \
    return SQL_ERROR;\
}
```

```
}

/* prepare + execute SQL statement */
int execute_cmd(SQLCHAR *sql)
{
    if ( NULL == sql )
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLPrepare(h_stmt, sql, SQL_NTS))
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

/* execute + commit handle */
int commit_exec()
{
    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    /* Manual commit */
    if ( SQL_SUCCESS != SQLEndTran(SQL_HANDLE_DBC, h_conn, SQL_COMMIT))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

int begin_unit_test()
{
    SQLINTEGER  ret;

    /* Allocate an environment handle. */
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &h_env);
    if ((SQL_SUCCESS != ret) && (SQL_SUCCESS_WITH_INFO != ret))
    {
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_ENV failed ! ret = %d", ret);
        return SQL_ERROR;
    }

    /* Set the version number before connection. */
    if (SQL_SUCCESS != SQLSetEnvAttr(h_env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3,
0))
    {
        print_error();
        printf("\n begin_unit_test::SQLSetEnvAttr SQL_ATTR_ODBC_VERSION failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Allocate a connection handle. */
    ret = SQLAllocHandle(SQL_HANDLE_DBC, h_env, &h_conn);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_DBC failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
    }
}
```

```
        return SQL_ERROR;
    }

    /* Establish a connection. */
    ret = SQLConnect(h_conn, (SQLCHAR*) "gaussdb", SQL_NTS,
                    (SQLCHAR*) NULL, 0, NULL, 0);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLConnect failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Allocate a statement handle. */
    ret = SQLAllocHandle(SQL_HANDLE_STMT, h_conn, &h_stmt);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_STMT failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

void end_unit_test()
{
    /* Release a statement handle. */
    if (NULL != h_stmt)
    {
        SQLFreeHandle(SQL_HANDLE_STMT, h_stmt);
    }

    /* Release a connection handle. */
    if (NULL != h_conn)
    {
        SQLDisconnect(h_conn);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
    }

    /* Release an environment handle. */
    if (NULL != h_env)
    {
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
    }

    return;
}

int main()
{
    /* begin test */
    if (begin_unit_test() != SQL_SUCCESS)
    {
        printf("\n begin_test_unit failed.");
        return SQL_ERROR;
    }
    /* The handle configuration is the same as that in the preceding case. */
    int i = 0;
    SQLCHAR* sql_drop = "drop table if exists test_bindnumber_001";
    SQLCHAR* sql_create = "create table test_bindnumber_001("
                           "f4 number, f5 number(10, 2)";
    SQLCHAR* sql_insert = "insert into test_bindnumber_001 values(?, ?)";
    SQLCHAR* sql_select = "select * from test_bindnumber_001";
}
```

```
SQLLEN    RowCount;
SQL_NUMERIC_STRUCT st_number;
SQLCHAR   getValue[2][MESSAGE_BUFFER_LEN];

/* Step 1. Create a table. */
RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
RETURN_IF_NOT_SUCCESS(execute_cmd(sql_create));

/* Step 2.1 Bind parameters using the SQL_NUMERIC_STRUCT structure. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));

/* First line: 1234.5678 */
memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
st_number.precision = 8;
st_number.scale = 4;
st_number.sign = 1;
st_number.val[0] = 0x4E;
st_number.val[1] = 0x61;
st_number.val[2] = 0xBC;

RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));

/* Disable the automatic commit function. */
SQLSetConnectAttr(h_conn, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Second line: 12345678 */
memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
st_number.precision = 8;
st_number.scale = 0;
st_number.sign = 1;
st_number.val[0] = 0x4E;
st_number.val[1] = 0x61;
st_number.val[2] = 0xBC;

RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Third line: 12345678 */
memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
st_number.precision = 0;
st_number.scale = 4;
st_number.sign = 1;
st_number.val[0] = 0x4E;
st_number.val[1] = 0x61;
st_number.val[2] = 0xBC;

RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Step 2.2 Bind parameters by using the SQL_C_CHAR character string in the fourth line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
```

```
SQLCHAR*  szNumber = "1234.5678";
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Step 2.3 Bind parameters by using SQL_C_FLOAT in the fifth line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
SQLREAL  fNumber = 1234.5678;
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Step 2.4 Bind parameters by using SQL_C_DOUBLE in the sixth line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
SQLDOUBLE  dNumber = 1234.5678;
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

SQLBIGINT  bNumber1 = 0xFFFFFFFFFFFFFFFF;
SQLBIGINT  bNumber2 = 12345;

/* Step 2.5 Bind parameters by using SQL_C_SBIGINT in the seventh line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Step 2.6 Bind parameters by using SQL_C_UBIGINT in the eighth line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

SQLLEN  lNumber1 = 0xFFFFFFFFFFFFFFFF;
SQLLEN  lNumber2 = 12345;

/* Step 2.7 Bind parameters by using SQL_C_LONG in the ninth line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Step 2.8 Bind parameters by using SQL_C_ULONG in the tenth line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
```

```
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

SQLSMALLINT sNumber = 0xFFFF;

/* Step 2.9 Bind parameters by using SQL_C_SHORT in the eleventh line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Step 2.10 Bind parameters by using SQL_C_USHORT in the twelfth line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

SQLCHAR cNumber = 0xFF;

/* Step 2.11 Bind parameters by using SQL_C_TINYINT in the thirteenth line. */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Step 2.12 Bind parameters by using SQL_C_UTINYINT in the fourteenth line.*/
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(1, RowCount);

/* Use the character string type to unify the expectation. */
SQLCHAR* expectValue[14][2] = {"1234.5678", "1234.57"},
{"12345678", "12345678"},
{"0", "0"},
{"1234.5678", "1234.57"},
{"1234.5677", "1234.57"},
{"1234.5678", "1234.57"},
{"-1", "12345"},
{"18446744073709551615", "12345"},
{"-1", "12345"},
{"4294967295", "12345"},
{"-1", "-1"},
{"65535", "65535"},
{"-1", "-1"},
{"255", "255"},
};
```

```
RETURN_IF_NOT_SUCCESS(execute_cmd(sql_select));
while ( SQL_NO_DATA != SQLFetch(h_stmt))
{
    RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 1, SQL_C_CHAR, &getValue[0],
MESSAGE_BUFFER_LEN, NULL));
    RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 2, SQL_C_CHAR, &getValue[1],
MESSAGE_BUFFER_LEN, NULL));
    i++;
}
printf("\nComplete!\n");
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(i, RowCount);
SQLCloseCursor(h_stmt);
/* Final step. Drop the table and restore the environment. */
RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
RETURN_IF_NOT_SUCCESS(commit_exec());
end_unit_test();
}
```

NOTE

In the preceding example, the **number** column is defined. When the SQLBindParameter API is called, the performance of binding SQL_NUMERIC is higher than that of SQL_LONG. If char is used, the data type needs to be converted when data is inserted to the database server, causing a performance bottleneck.

The running result is as follows:

```
Complete!
```

5.5.4 ODBC Interface Reference

The ODBC interface is a set of API functions provided to users. This chapter describes its common interfaces. For details on other interfaces, see "ODBC Programmer's Reference" at MSDN ([https://msdn.microsoft.com/en-us/library/windows/desktop/ms714177\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms714177(v=vs.85).aspx)).

5.5.4.1 SQLAllocEnv

In ODBC 3.x, SQLAllocEnv (an ODBC 2.x function) was deprecated and replaced by SQLAllocHandle. For details, see [SQLAllocHandle](#).

5.5.4.2 SQLAllocConnect

In ODBC 3.x, SQLAllocConnect (an ODBC 2.x function) was deprecated and replaced by SQLAllocHandle. For details, see [SQLAllocHandle](#).

5.5.4.3 SQLAllocHandle

Description

Allocates environment, connection, statement, or descriptor handles. This function replaces the deprecated ODBC 2.x functions SQLAllocEnv, SQLAllocConnect, and SQLAllocStmt.

Prototype

```
SQLRETURN SQLAllocHandle(SQLSMALLINT HandleType,
SQLHANDLE InputHandle,
SQLHANDLE *OutputHandlePtr);
```

Parameters

Table 5-41 SQLAllocHandle parameters

Keyword	Description
HandleType	<p>Type of handle to be allocated by SQLAllocHandle. The value must be one of the following:</p> <ul style="list-style-type: none"> • SQL_HANDLE_ENV (environment handle) • SQL_HANDLE_DBC (connection handle) • SQL_HANDLE_STMT (statement handle) • SQL_HANDLE_DESC (descriptor handle) <p>The handle allocation sequence is: environment handle > connection handle > statement handle. A later allocated handle depends on a previously allocated handle.</p>
InputHandle	<p>Existing handle to use as a context for the new handle being allocated. The InputHandle parameter specifies the parent handle of the handle to be created to establish the hierarchical relationship between handles. Handles of different types have different hierarchical relationships. The InputHandle parameter is used to specify the relationship.</p> <ul style="list-style-type: none"> • If HandleType is SQL_HANDLE_ENV, the value is SQL_NULL_HANDLE, indicating that there is no parent handle. • If HandleType is SQL_HANDLE_DBC, this must be an environment handle, indicating that the connection handle is created in this environment. • If HandleType is SQL_HANDLE_STMT or SQL_HANDLE_DESC, it must be a connection handle, indicating that the statement handle is created under the connection.
OutputHandlePtr	<p>Output parameter: OutputHandlePtr is a pointer that points to the SQLHANDLE type and is used to return a new handle allocated.</p>

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLAllocHandle returns **SQL_ERROR** when it is used to allocate a non-environment handle, it sets **OutputHandlePtr** to **SQL_NULL_HDBC**, **SQL_NULL_HSTMT**, or **SQL_NULL_HDESC**. The application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to the value of **InputHandle**, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.4 SQLAllocStmt

In ODBC 3.x, SQLAllocStmt was deprecated and replaced by SQLAllocHandle. For details, see [SQLAllocHandle](#).

5.5.4.5 SQLBindCol

Description

Binds application data buffers to columns in a result set.

Prototype

```
SQLRETURN SQLBindCol(SQLHSTMT StatementHandle,
    SQLUSMALLINT ColumnNumber,
    SQLSMALLINT TargetType,
    SQLPOINTER TargetValuePtr,
    SQLLEN BufferLength,
    SQLLEN *StrLen_or_IndPtr);
```

Parameters

Table 5-42 SQLBindCol parameters

Keyword	Description
StatementHandle	Statement handle.
ColumnNumber	Number of the column to be bound. The column number starts with 0 and increases in ascending order. Column 0 is the bookmark column. If no bookmark column is set, column numbers start with 1.
TargetType	C data type in the buffer.
TargetValuePtr	Output parameter: pointer to the buffer bound with the column. The SQLFetch function returns data in the buffer. If this parameter is a null pointer, StrLen_or_IndPtr is a valid value.
BufferLength	Length of the buffer pointed to by TargetValuePtr , in bytes.

Keyword	Description
StrLen_or_IndPtr	Output parameter: pointer to the length or indicator of the buffer. If the value is null, no length or indicator is used.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLBindCol returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.6 SQLBindParameter

Description

Binds parameter markers in an SQL statement to a buffer.

Prototype

```
SQLRETURN SQLBindParameter(SQLHSTMT StatementHandle,
    SQLUSMALLINT ParameterNumber,
    SQLSMALLINT InputOutputType,
    SQLSMALLINT ValuetType,
    SQLSMALLINT ParameterType,
    SQLULEN ColumnSize,
    SQLSMALLINT DecimalDigits,
    SQLPOINTER ParameterValuePtr,
    SQLLEN BufferLength,
    SQLLEN *StrLen_or_IndPtr);
```

Parameters

Table 5-43 SQLBindParameter parameters

Keyword	Description
StatementHandle	Statement handle.
ParameterNumber	Parameter marker number, starting with 1 and increasing in ascending order.
InputOutputType	Input/output type of the parameter. The value can be SQL_PARAM_INPUT , SQL_PARAM_OUTPUT or SQL_PARAM_INPUT_OUTPUT .
ValueType	C data type of the parameter. The value can be SQL_C_CHAR , SQL_C_LONG or SQL_C_DOUBLE .
ParameterType	SQL data type of the parameter. The value can be SQL_CHAR , SQL_INTEGER , or SQL_DOUBLE .
ColumnSize	Size of the column or expression marked by the corresponding parameter.
DecimalDigits	Decimal number of the column or expression marked by the corresponding parameter.
ParameterValuePtr	Pointer to the storage parameter buffer.
BufferLength	Size of the ParameterValuePtr buffer in bytes.
StrLen_or_IndPtr	Pointer to the length or indicator of the buffer. If the value is null, no length or indicator is used.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLBindParameter returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.7 SQLColAttribute

Description

Returns the descriptor information about a column in the result set.

Prototype

```
SQLRETURN SQLColAttribute(SQLHSTMT StatementHandle,
    SQLUSMALLINT ColumnNumber,
    SQLUSMALLINT FieldIdentifier,
    SQLPOINTER CharacterAttributePtr,
    SQLSMALLINT BufferLength,
    SQLSMALLINT *StringLengthPtr,
    SQLLEN *NumericAttributePtr);
```

Parameters

Table 5-44 SQLColAttribute parameters

Keyword	Description
StatementHandle	Statement handle.
ColumnNumber	Column number of the field to be queried, starting with 1 and increasing in ascending order.
FieldIdentifier	Field identifier of ColumnNumber in IRD.
CharacterAttributePtr	Output parameter: pointer to the buffer that returns the FieldIdentifier value.
BufferLength	<ul style="list-style-type: none"> Indicates the length of the buffer if FieldIdentifier is an ODBC-defined field and CharacterAttributePtr points to a string or a binary buffer. Ignore this parameter if FieldIdentifier is an ODBC-defined field and CharacterAttributePtr points to an integer.
StringLengthPtr	Output parameter: pointer to a buffer in which the total number of valid bytes (for string data) is stored in *CharacterAttributePtr . Ignore the value of BufferLength if the data is not a string.
NumericAttributePtr	Output parameter: pointer to an integer buffer in which the value of FieldIdentifier in the ColumnNumber row of the IRD is returned.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If `SQLColAttribute` returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.8 SQLConnect

Description

Establishes a connection between a driver and a data source. After the connection is established, the connection handle can be used to access all information about the data source, including its application operating status, transaction processing status, and error information.

Prototype

```
SQLRETURN SQLConnect(SQLHDBC ConnectionHandle,  
SQLCHAR *ServerName,  
SQLSMALLINT NameLength1,  
SQLCHAR *UserName,  
SQLSMALLINT NameLength2,  
SQLCHAR *Authentication,  
SQLSMALLINT NameLength3);
```

Parameters

Table 5-45 SQLConnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from <code>SQLAllocHandle</code> .
ServerName	Name of the data source to connect.
NameLength1	Length of ServerName .

Keyword	Description
UserName	Username of the database in the data source.
NameLength2	Length of UserName .
Authentication	User password of the database in the data source.
NameLength3	Length of Authentication .

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLConnect returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.9 SQLDisconnect

Description

Closes the connection associated with a database connection handle.

Prototype

```
SQLRETURN SQLDisconnect(SQLHDBC ConnectionHandle);
```

Parameters

Table 5-46 SQLDisconnect parameter

Keyword	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If `SQLDisconnect` returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.10 SQLExecDirect

Description

Executes a prepared statement specified in this parameter. `SQLExecDirect` is the fastest method for executing only one SQL statement at a time.

Prototype

```
SQLRETURN SQLExecDirect(SQLHSTMT StatementHandle,
                        SQLCHAR *StatementText,
                        SQLINTEGER TextLength);
```

Parameters

Table 5-47 SQLExecDirect parameters

Keyword	Description
StatementHandle	Statement handle, obtained from <code>SQLAllocHandle</code> .
StatementText	SQL statement to be executed. Multiple statements cannot be executed at a time.
TextLength	Length of StatementText .

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_NEED_DATA** indicates that parameters provided before executing the SQL statement are insufficient.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.

Precautions

If `SQLExecDirect` returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call `SQLGetDiagRec`, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.11 SQLExecute

Description

If a statement contains a parameter marker, the `SQLExecute` function uses the current value of the parameter marker to execute a prepared SQL statement.

Prototype

```
SQLRETURN SQLExecute(SQLHSTMT StatementHandle);
```

Parameters

Table 5-48 SQLExecute parameter

Keyword	Description
StatementHandle	Statement handle to be executed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_NEED_DATA** indicates that parameters provided before executing the SQL statement are insufficient.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If `SQLExecute` returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.12 SQLFetch

Description

Advances the cursor to the next row of the result set and retrieves any bound columns.

Prototype

```
SQLRETURN SQLFetch(SQLHSTMT StatementHandle);
```

Parameters

Table 5-49 SQLFetch parameter

Keyword	Description
StatementHandle	Statement handle, obtained from <code>SQLAllocHandle</code> .

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLFetch returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.13 SQLFreeStmt

In ODBC 3.x, SQLFreeStmt (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see [SQLFreeHandle](#).

5.5.4.14 SQLFreeConnect

In ODBC 3.x, SQLFreeConnect (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see [SQLFreeHandle](#).

5.5.4.15 SQLFreeHandle

Description

Releases resources associated with a specific environment, connection, or statement handle. It replaces the ODBC 2.x functions: SQLFreeEnv, SQLFreeConnect, and SQLFreeStmt.

Prototype

```
SQLRETURN SQLFreeHandle(SQLSMALLINT HandleType,
                        SQLHANDLE Handle);
```

Parameters

Table 5-50 SQLFreeHandle parameters

Keyword	Description
HandleType	Type of handle to be freed by SQLFreeHandle. The value must be one of the following: <ul style="list-style-type: none"> • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC If none of them is the value of HandleType , SQLFreeHandle returns SQL_INVALID_HANDLE .

Keyword	Description
Handle	Name of the handle to be freed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLFreeHandle returns **SQL_ERROR**, the handle is still valid.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.16 SQLFreeEnv

In ODBC 3.x, SQLFreeEnv (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see [SQLFreeHandle](#).

5.5.4.17 SQLPrepare

Description

Prepares an SQL statement to be executed.

Note that ODBC does not support the kernel reuse plan when sending prepared statements. As a result, a new plan needs to be generated for each execution, causing high CPU usage. If services have requirements on plan reuse, you are advised to use the JDBC client.

Prototype

```
SQLRETURN SQLPrepare(SQLHSTMT StatementHandle,  
SQLCHAR *StatementText,  
SQLINTEGER TextLength);
```

Parameters

Table 5-51 SQLPrepare parameters

Keyword	Description
StatementHandle	Statement handle.
StatementText	SQL text string.
TextLength	Length of StatementText .

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLPrepare returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.18 SQLGetData

Description

SQLGetData is used to retrieve data for a single column in the result set. It can be called multiple times to partially retrieve variable-length data.

Prototype

```
SQLRETURN SQLGetData(SQLHSTMT StatementHandle,  
SQLUSMALLINT Col_or_Param_Num,  
SQLSMALLINT TargetType,  
SQLPOINTER TargetValuePtr,  
SQLLEN BufferLength,  
SQLLEN *StrLen_or_IndPtr);
```

Parameters

Table 5-52 SQLGetData parameters

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.
Col_or_Param_Num	Column number for which the data retrieval is requested. The column number starts with 1 and increases in ascending order. The number of the bookmark column is 0.
TargetType	C data type in the TargetValuePtr buffer. If TargetType is SQL_ARD_TYPE , the driver uses the data type of the SQL_DESC_CONCISE_TYPE column in ARD. If it is SQL_C_DEFAULT , the driver selects a default data type according to the source SQL data type.
TargetValuePtr	Output parameter: pointer to the pointer that points to the buffer where the data is located.
BufferLength	Size of the buffer pointed to by TargetValuePtr .
StrLen_or_IndPtr	Output parameter: pointer to the buffer where the length or identifier value is returned.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLGetData returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.19 SQLGetDiagRec

Description

Returns the current values of multiple fields in a diagnostic record that contains error, warning, and status information.

Prototype

```
SQLRETURN SQLGetDiagRec(SQLSMALLINT HandleType,
                        SQLHANDLE Handle,
                        SQLSMALLINT RecNumber,
                        SQLCHAR *SQLState,
                        SQLINTEGER *NativeErrorPtr,
                        SQLCHAR *MessageText,
                        SQLSMALLINT BufferLength,
                        SQLSMALLINT *TextLengthPtr);
```

Parameters

Table 5-53 SQLGetDiagRec parameters

Keyword	Description
HandleType	Handle-type identifier that describes the type of handle for which diagnostics are desired. The value must be one of the following: <ul style="list-style-type: none"> • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
Handle	Handle for the diagnostic data structure. Its type is indicated by HandleType . If HandleType is set to SQL_HANDLE_ENV , Handle may indicate a shared or non-shared environment handle.
RecNumber	Status record from which the application seeks information. Status records are numbered from 1.
SQLState	Output parameter: pointer to a buffer that saves the 5-character SQLSTATE code pertaining to RecNumber .
NativeErrorPtr	Output parameter: pointer to a buffer that saves the native error code.
MessageText	Pointer to a buffer that saves text strings of diagnostic information.
BufferLength	Length of MessageText .

Keyword	Description
TextLengthPtr	Output parameter: pointer to the buffer, the total number of bytes in the returned MessageText . If the number of bytes available to return is greater than BufferLength , then the diagnostics information text in MessageText is truncated to BufferLength minus the length of the null termination character.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

SQLGetDiagRec does not release diagnostic records for itself. It uses the following return values to report execution results:

- **SQL_SUCCESS** indicates that the function successfully returns diagnostic information.
- **SQL_SUCCESS_WITH_INFO** indicates that the **MessageText** buffer is too small to hold the requested diagnostic information. No diagnostic records are generated.
- **SQL_INVALID_HANDLE** indicates that the handle indicated by **HandleType** and **Handle** is an invalid handle.
- **SQL_ERROR** indicates that the value of **RecNumber** is less than or equal to 0 or the value of **BufferLength** is less than 0.

If an ODBC function returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, you can call SQLGetDiagRec to obtain the **SQLSTATE** value. [Table 2 SQLSTATE values](#) lists the possible **SQLSTATE** values.

Table 5-54 SQLSTATE values

SQLSTATE	Error	Description
HY000	General error.	An error occurs when there is no specific SQLSTATE.
HY001	Memory allocation error.	The driver is unable to allocate memory required to support execution or completion of the function.

SQLSTATE	Error	Description
HY008	Operation canceled.	SQLCancel is called to terminate the statement execution, but the StatementHandle function is still called.
HY010	Function sequence error.	The function is called prior to sending data to data parameters or columns being executed.
HY013	Memory management error.	The function fails to be called. The error may be caused by low memory conditions.
HYT01	Connection timeout.	The timeout period expired before the application was able to connect to the data source.
IM001	Function not supported by the driver.	The called function is not supported by the StatementHandle driver.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.20 SQLSetConnectAttr

Description

Sets connection attributes.

Prototype

```
SQLRETURN SQLSetConnectAttr(SQLHDBC ConnectionHandle,
                             SQLINTEGER Attribute,
                             SQLPOINTER ValuePtr,
                             SQLINTEGER StringLength);
```

Parameters

Table 5-55 SQLSetConnectAttr parameters

Keyword	Description
ConnectionHandle	Connection handle.
Attribute	Attribute to set.

Keyword	Description
ValuePtr	Pointer to the Attribute value. ValuePtr depends on the Attribute value, and is a 32-bit unsigned integer value or a pointer to a null-terminated string, a binary buffer, or a driver-specified value. If the ValuePtr parameter is driver-specific value, it may be a signed integer.
StringLength	If ValuePtr points to a string or a binary buffer, StringLength is the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If `SQLSetConnectAttr` returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call `SQLGetDiagRec`, with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.21 SQLSetEnvAttr

Description

Sets environment attributes.

Prototype

```
SQLRETURN SQLSetEnvAttr(SQLHENV EnvironmentHandle,
                        SQLINTEGER Attribute,
                        SQLPOINTER ValuePtr,
                        SQLINTEGER StringLength);
```

Parameters

Table 5-56 SQLSetEnvAttr parameters

Keyword	Description
EnvironmentHandle	Environment handle.
Attribute	Environment attribute to be set. The value must be one of the following: <ul style="list-style-type: none">• SQL_ATTR_ODBC_VERSION: ODBC version.• SQL_CONNECTION_POOLING: connection pool attribute.• SQL_OUTPUT_NTS: string type returned by the driver.
ValuePtr	Pointer to the Attribute value. ValuePtr depends on the Attribute value, and can be a 32-bit integer value or a null-terminated string.
StringLength	If ValuePtr points to a string or a binary buffer, StringLength is the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLSetEnvAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_ENV** and **EnvironmentHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

For details, see [Typical Application Development Examples](#).

5.5.4.22 SQLSetStmtAttr

Description

Sets attributes related to a statement.

Prototype

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT StatementHandle,
                          SQLINTEGER Attribute,
                          SQLPOINTER ValuePtr,
                          SQLINTEGER StringLength);
```

Parameters

Table 5-57 SQLSetStmtAttr parameters

Keyword	Description
StatementHandle	Statement handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the Attribute value. ValuePtr depends on the Attribute value, and can be a 32-bit unsigned integer value or a pointer to a null-terminated string, a binary buffer, or a driver-specified value. If the ValuePtr parameter is driver-specific value, it may be a signed integer.
StringLength	If ValuePtr points to a string or a binary buffer, StringLength is the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLSetStmtAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call [SQLGetDiagRec](#), with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

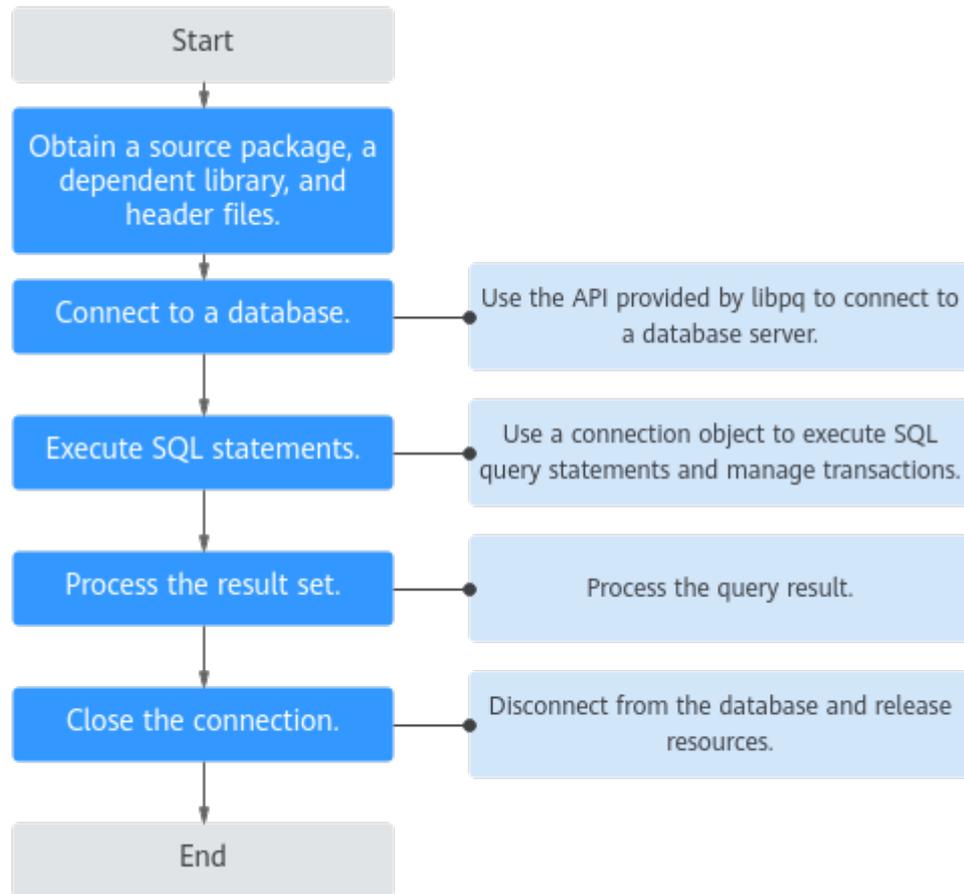
For details, see [Typical Application Development Examples](#).

5.6 Development Based on libpq

5.6.1 Development Process

Figure 5-4 shows the libpq-based development process.

Figure 5-4 libpq-based development process



5.6.2 Development Process

5.6.2.1 Obtaining a Driver Package, a Dependency Library, and Header Files

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-58](#).

Table 5-58 Driver package download list

Version	Download Address
V2.0-3.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

- Verifying the software package integrity on a Linux server:
 - a. Upload the software package and verification package to the same directory on the VM.
 - b. Run the following command to verify the integrity of the software package:

```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

If **OK** is displayed in the command output, the verification is successful.

```
GaussDB_driver.zip: OK
```
- Verifying the software package integrity on a Windows server:
 - a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
 - b. Run the following command to obtain the hash value of the driver package:

```
certutil -hashfile {local_directory_of_the_driver_package}\{driver_package_name} sha256
```

 - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
 - Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **GaussDB_driver.zip**.Example: **certutil -hashfile C:\Users\GaussDB_driver.zip sha256**
 - c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-58**.
 - If they are consistent, the verification is successful.
 - If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Obtain the library and header files of libpq from **GaussDB-Kernel_Database version number_OS information_64bit_Libpq.tar.gz** in the driver package. The required header file is stored in the **include** folder, and the **lib** folder contains the required libpq library file. Programs that use libpq must include the header file **libpq-fe.h** and must connect to the libpq library.

NOTE

In addition to **libpq-fe.h**, the **include** folder contains the header files **postgres_ext.h**, **gs_thread.h**, and **gs_threadlocal.h** by default. These three header files are the dependency files of **libpq-fe.h**.

5.6.2.2 Connecting to a Database

Connecting to a database is the first step in developing an application using libpq. In this case, you can use functions, such as **PQconnectdb** and **PQsetdbLogin**, to connect to the database server. These functions return a connection object. You need to save the connection object for subsequent database operations.

The following uses the development source program **testlibpq.c** as an example (for details about the complete example, see [Establishing a Database Connection, Executing SQL Statements, and Returning Results](#)).

```
/*
 * Note: testlibpq.c source program provides basic and common application scenarios of libpq.
 * The PQconnectdb, PQexec, PQntuples, and PQfinish APIs provided by libpq are used to establish database
 * connections, execute SQL statements, obtain returned results, and clear resources.
 * Header file. For details about user-defined functions, see the complete example.
 */

/* The values of variables such as user and passwd must be read from environment variables or
 * configuration files. Environment variables need to be configured as required. If no environment variable is
 * used, a character string can be directly assigned. */
const char conninfo[1024];
PGconn *conn;
PGresult *res;
int nFields;
int i,j;
char *passwd = getenv("EXAMPLE_PASSWD_ENV");
char *port = getenv("EXAMPLE_PORT_ENV");
char *host = getenv("EXAMPLE_HOST_ENV");
char *username = getenv("EXAMPLE_USERNAME_ENV");
char *dbname = getenv("EXAMPLE_DBNAME_ENV");

/*
 * This value is used when the user provides the value of the conninfo character string in the command line.
 * Otherwise, the environment variables or all other connection parameters
 * use the default values.
 */
if (argc > 1)
    strcpy(conninfo, argv[1]);
else
    sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
            password=%s",
            dbname, port, host, username, passwd);

/* Connect to the database. */
conn = PQconnectdb(conninfo);

/* Check whether the backend connection has been successfully established. */
if (PQstatus(conn) != CONNECTION_OK)
{
    fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
    exit_nicely(conn);
}
```

5.6.2.3 Running SQL Statements

You can use the function **PQexec** to execute SQL query statements through the connection object. This includes querying data (SELECT), inserting data, updating data, and deleting data. If multiple SQL statements are executed concurrently as a transaction, use the transaction control function. For example, run SQL statements such as BEGIN, COMMIT, and ROLLBACK to control the start, committing, and rollback of a transaction. Handle the errors after executing the SQL query statements.

The following is an example (for details about the complete example, see [Establishing a Database Connection, Executing SQL Statements, and Returning Results](#)):

```
/*
 * After the connection is successful
```

```

* Since a cursor is used in the test case, a transaction block is required.
* Put all data in one "select * from pg_database".
* PQexec() is too simple and is not recommended.
*/

/* Start a transaction block. */
res = PQexec(conn, "BEGIN");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/*
* PQclear PGresult should be executed when it is no longer needed, to avoid memory leakage.
*/
PQclear(res);

/*
* Fetch data from the pg_database system catalog.
*/
res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
PQclear(res);

res = PQexec(conn, "FETCH ALL in myportal");
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

```

5.6.2.4 Processing Data in a Result Set

libpq provides functions, such as [PQnfields](#) and [PQfname](#), to help you properly parse and process the results of queries by using SELECT.

The following is an example (for details about the complete example, see [Establishing a Database Connection, Executing SQL Statements, and Returning Results](#)):

```

/* First, print out the attribute name. */
nFields = PQnfields(res);
for (i = 0; i < nFields; i++)
    printf("%-15s", PQfname(res, i));
printf("\n\n");

/* Print lines. */
for (i = 0; i < PQntuples(res); i++)
{
    for (j = 0; j < nFields; j++)
        printf("%-15s", PQgetvalue(res, i, j));
    printf("\n");
}

/* Release the memory of the result object to avoid memory leakage. */
PQclear(res);

```

5.6.2.5 Closing a Connection

libpq usually uses the function **PQfinish** to close the connection to the database. If the application has established multiple connections, make sure that each connection is closed correctly.

The following is an example (for details about the complete example, see [Establishing a Database Connection, Executing SQL Statements, and Returning Results](#)):

```
/* Close the portal. We do not need to check for errors. */
res = PQexec(conn, "CLOSE myportal");
PQclear(res);

/* End the transaction. */
res = PQexec(conn, "END");
PQclear(res);

/* Close the database connection and clean up the database. */
PQfinish(conn);
```

5.6.3 Typical Application Development Examples

5.6.3.1 Establishing a Database Connection, Executing SQL Statements, and Returning Results

NOTE

To compile the libpq source program by running **gcc**, use the **-I** *directory* option to provide the installation location of the header file. (Sometimes the compiler looks for the default directory, so this option can be ignored.) Example:

```
gcc -I (Directory where header files are located) -L (Directory where the libpq library is located) -o
testlibpq testlibpq.c -lpq
```

Run the following command:

```
./testlibpq.c
```

If the makefile is used, add the following option to variables **CPPFLAGS**, **LDFLAGS**, and **LIBS**:

```
CPPFLAGS += -I (Directory of the header file)
LDFLAGS += -L (Directory of the libpq library)
LIBS += -lpq
```

Example:

```
CPPFLAGS += -I$(GAUSSHOME)/include/libpq
LDFLAGS += -L$(GAUSSHOME)/lib
```

```
/*
 * testlibpq.c
 * Note: testlibpq.c source program provides basic and common application scenarios of libpq.
 * The PQconnectdb, PQexec, PQntuples, and PQfinish APIs provided by libpq are used to establish database
 * connections, execute SQL statements, obtain returned results, and clear resources.
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main(int argc, char **argv)
{
```

```
/* The values of variables such as user and passwd must be read from environment variables or
configuration files. Environment variables need to be configured as required. If no environment variable is
used, a character string can be directly assigned. */
const char conninfo[1024];
PGconn *conn;
PGresult *res;
int nFields;
int i,j;
char *passwd = getenv("EXAMPLE_PASSWD_ENV");
char *port = getenv("EXAMPLE_PORT_ENV");
char *host = getenv("EXAMPLE_HOST_ENV");
char *username = getenv("EXAMPLE_USERNAME_ENV");
char *dbname = getenv("EXAMPLE_DBNAME_ENV");

/*
 * This value is used when the user provides the value of the conninfo character string in the command
line.
 * Otherwise, the environment variables or the default values
 * are used for all other connection parameters.
 */
if (argc > 1)
    strcpy(conninfo, argv[1]);
else
    sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, host, username, passwd);

/* Connect to the database. */
conn = PQconnectdb(conninfo);

/* Check whether the backend connection has been successfully established. */
if (PQstatus(conn) != CONNECTION_OK)
{
    fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
    exit_nicely(conn);
}

/*
 * After the connection is successful
 * Since a cursor is used in the test case, a transaction block is required.
 * Put all data in one "select * from pg_database".
 * PQexec() is too simple and is not recommended.
 */

/* Start a transaction block. */
res = PQexec(conn, "BEGIN");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/*
 * PQclear PGresult should be executed when it is no longer needed, to avoid memory leakage.
 */
PQclear(res);

/*
 * Fetch data from the pg_database system catalog.
 */
res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
```

```

}
PQclear(res);

res = PQexec(conn, "FETCH ALL in myportal");
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/* Print out the attribute name. */
nFields = PQnfields(res);
for (i = 0; i < nFields; i++)
    printf("%-15s", PQfname(res, i));
printf("\n\n");

/* Print lines. */
for (i = 0; i < PQntuples(res); i++)
{
    for (j = 0; j < nFields; j++)
        printf("%-15s", PQgetvalue(res, i, j));
    printf("\n");
}

/* Release the memory of the result object to avoid memory leakage. */
PQclear(res);

/* Close the portal. Do not check for errors. */
res = PQexec(conn, "CLOSE myportal");
PQclear(res);

/* End the transaction. */
res = PQexec(conn, "END");
PQclear(res);

/* Close the database connection and clean up the database. */
PQfinish(conn);

return 0;
}

```

In the following command output, **user_name** indicates the username of a database administrator, which varies according to the actual environment:

datname	datdba	encoding	datcollate	datctype	datistemplate	dataallowconn	datconnlimit	datlastsysoid	datfrozenxid	dattablespace	datcompatibilitydatacl	datfrozenxid64	datminmxid	dattimezone	dattype
template_pdb	10	7	en_US.UTF-8	en_US.UTF-8	t	t	-1								
12837	0	1663	A	3	2	PRC	P								
templatea	10	7	en_US.UTF-8	en_US.UTF-8	t	f	-1								
12837	0	1663	A	{=c/user_name,user_name=CTc/user_name}			41372								
2	PRC	D													
template1	10	7	en_US.UTF-8	en_US.UTF-8	t	t	-1								
12837	0	1663	A	{=c/user_name,user_name=CTc/user_name}			40414								
2	PRC	D													
templatem	10	7	en_US.UTF-8	en_US.UTF-8	t	t	-1								
12837	0	1663	M	{=c/user_name,user_name=CTc/user_name}			55146								
2	PRC	D													
template0	10	7	en_US.UTF-8	en_US.UTF-8	t	f	-1								
12837	0	1663	A	{=c/user_name,user_name=CTc/user_name}			39935								
2	PRC	D													
postgres	10	7	en_US.UTF-8	en_US.UTF-8	f	t	-1								
12837	0	1663	A	40893	2	PRC	D								

5.6.3.2 Executing Prepared Statements

```
/*
 * testlibpq2.c Test PQprepare
 * PQprepare creates a prepared statement with specified parameters for PQexecPrepared to execute the
 prepared statement.
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
    /* The values of variables such as user and passwd must be read from environment variables or
 configuration files. Environment variables need to be configured as required. If no environment variable is
 used, a character string can be directly assigned. */
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /* Use PQconnectdb to connect to the database. The detailed connection information is as follows:
 connstr. */
    sprintf(connstr,
            "hostaddr=%s dbname=%s port=%s user=%s password=%s",
            hostaddr, dbname, port, username, passwd);
    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
        return -1;
    }

    /* Create table t01. */
    res = PQexec(conn, "DROP TABLE IF EXISTS t01;CREATE TABLE t01(a int, b int);INSERT INTO t01
 values(1, 23);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        printf("Command failed: %s.\n", PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }

    /* cmd_sql query */
    sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
    /* Parameter corresponding to $1 in cmd_sql */
    paramTypes[0] = 23;
    /* PQprepare creates a prepared statement with given parameters. */
    res = PQprepare(conn,
                   "pre_name",
                   cmd_sql,
```

```
        1,
        paramTypes);
if( PQresultStatus(res) != PGRES_COMMAND_OK )
{
    printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
    PQfinish(conn);
    return -1;
}
PQclear(res);
paramValues[0] = cid;
for (k=0; k<2; k++)
{
    sprintf(cid, "%d", 1);
    paramLengths[0] = 6;
    paramFormats[0] = 0;
    /* Execute the prepared statement. */
    res = PQexecPrepared(conn,
                        "pre_name",
                        1,
                        paramValues,
                        paramLengths,
                        paramFormats,
                        0);
    if( (PQresultStatus(res) != PGRES_COMMAND_OK) && (PQresultStatus(res) != PGRES_TUPLES_OK))
    {
        printf("%s\n",PQerrorMessage(conn));
        PQclear(res);
        PQfinish(conn);
        return -1;
    }
    cnt = PQntuples(res);
    printf("return %d rows\n", cnt);
    for (i=0; i<cnt; i++)
    {
        printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
    }
    PQclear(res);
}
/* The execution is complete. Close the connection. */
PQfinish(conn);
return 0;
}
```

The command output is as follows:

```
Connect database success!
NOTICE: table "t01" does not exist, skipping
return 1 rows
row 0: 23
return 1 rows
row 0: 23
```

5.6.3.3 Binding Parameters and Returning a Binary Result

```
/*
 * testlibpq3.c
 * Test PQexecParams.
 * PQexecParams executes a command with parameters and requests the query result in binary format.
 * Before running this example, run the following command to populate a database:
 *
 *
 * CREATE TABLE test1 (i int4, t text);
 *
 * INSERT INTO test1 values (2, 'ho there');
 *
 * The expected output is as follows:
 *
 *
 * tuple 0: got
```

```
* i = (4 bytes) 2
* t = (8 bytes) 'ho there'
*
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * This function is used to print out the query results. The results are in binary format
 * and fetched from the table created in the comment above.
 */
static void
show_binary_results(PGresult *res)
{
    int    i;
    int    i_fnum,
          t_fnum;

    /* Use PQfnumber to avoid assumptions about field order in the result. */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char    *iptr;
        char    *tptr;
        int     ival;

        /* Obtain the field value. (Ignore the possibility that they may be null.) */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * The binary representation of INT4 is the network byte order,
         * which is better to be replaced with the local byte order.
         */
        ival = ntohl(*(uint32_t *) iptr);

        /*
         * The binary representation of TEXT is text, so libpq can append a zero byte to it,
         * and think of it as a C string.
         */

        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
              PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) %s\n",
              PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
```

```
{
    /* The values of variables such as user and passwd must be read from environment variables or
    configuration files. Environment variables need to be configured as required. If no environment variable is
    used, a character string can be directly assigned. */
    const char conninfo[1024];
    PGconn *conn;
    PGresult *res;
    const char *paramValues[1];
    int paramLengths[1];
    int paramFormats[1];
    uint32_t binaryIntVal;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
    * If the user provides a parameter on the command line,
    * The value of this parameter is a conninfo character string. Otherwise,
    * Use environment variables or default values.
    */
    if (argc > 1)
        strcpy(conninfo, argv[1]);
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, hostaddr, username, passwd);

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);

    /* Check whether the connection to the server was successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "drop table if exists test1;CREATE TABLE test1 (i int4, t text);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    res = PQexec(conn, "INSERT INTO test1 values (2, 'ho there');");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    /* Convert the integer value "2" to the network byte order. */
    binaryIntVal = htonl((uint32_t) 2);

    /* Set the parameter array for PQexecParams. */
    paramValues[0] = (char *) &binaryIntVal;
    paramLengths[0] = sizeof(binaryIntVal);
    paramFormats[0] = 1; /* Binary */
}
```

```
/* PQexecParams executes a command with parameters. */
res = PQexecParams(conn,
    "SELECT * FROM test1 WHERE i = $1::int4",
    1, /* One parameter */
    NULL, /* Enable the backend to deduce the parameter type. */
    paramValues,
    paramLengths,
    paramFormats,
    1); /* Binary result is required. */

if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
/* Output the binary result.*/
show_binary_results(res);

PQclear(res);

/* Close the database connection and clean up the database. */
PQfinish(conn);

return 0;
}
```

The command output is as follows:

```
tuple 0: got
i = (4 bytes) 2
t = (8 bytes) 'ho there'
```

5.6.4 libpq Interface Reference

5.6.4.1 Database Connection Control Functions

Database connection control functions control the connections to GaussDB servers. An application can connect to multiple servers at a time. For example, a client connects to multiple databases. Each connection is represented by a PGconn object, which is obtained from the function PQconnectdb, PQconnectdbParams, or PQsetdbLogin. Note that these functions will always return a non-null object pointer, unless memory allocation fails. The API for establishing a connection is stored in the PGconn object. The PQstatus function can be called to check the return value for a successful connection.

5.6.4.1.1 PQconnectdbParams

Description

Establishes a new connection with the database server.

Prototype

```
PGconn* PQconnectdbParams(const char* const* keywords, const char* const* values, int expand_dbname);
```

Parameters

Table 5-59 PQconnectdbParams parameters

Keyword	Description
keywords	An array of strings, each of which is a keyword.
values	Value assigned to each keyword.
expand_dbname	When expand_dbname is non-zero, the dbname keyword value can be recognized as a connection string. Only dbname that first appears is treated in this way, and any subsequent dbname value is treated as a database name.

Return Values

PGconn * points to the object pointer that contains a connection. The memory is allocated by the function internally.

Precautions

This function establishes a new database connection using the parameters taken from two NULL-terminated arrays. Unlike PQsetdbLogin, the parameter set can be extended without changing the function signature. Therefore, use of this function (or its non-blocking analogs PQconnectStartParams and PQconnectPoll) is preferred for new application programming.

5.6.4.1.2 PQconnectdb

Description

Establishes a new connection with the database server.

Prototype

```
PGconn* PQconnectdb(const char* conninfo);
```

Parameter

Table 5-60 PQconnectdb parameter

Keyword	Description
conninfo	Connection string. For details about the columns in the string, see section 5.6.4.5 .

Return Values

PGconn * points to the object pointer that contains a connection. The memory is allocated by the function internally.

Precautions

- This function opens a new connection with the parameters specified by **conninfo**.
- The input parameter can be empty, indicating that all default parameters can be used. It can contain one or more values separated by spaces or contain a URL.

Examples

For details, see [Typical Application Development Examples](#).

5.6.4.1.3 PQbackendPID

Supplementary Explanation

After GaussDB is multi-thread refactored based on PostgreSQL, the semantic of PQbackendPID is different from that in the native PostgreSQL libpq. In GaussDB, the return value of the PQbackendPID function indicates the slot ID of the background thread, not the backend PID of the background thread. Due to the preceding difference, you are not advised to execute this function by following the PostgreSQL semantics. To obtain the backend PID of the connection, you can use the `pg_backend_pid` system function. In addition, other driver APIs which depend on libpq and have the same names as PostgreSQL's APIs (such as the `get_backend_pid` function of the Python connection driver `psycopg2`) also comply with the preceding rule.

5.6.4.1.4 PQsetdbLogin

Description

Establishes a new connection with the database server.

Prototype

```
PGconn* PQsetdbLogin(const char* pghost, const char* pgport, const char* pgoptions, const char* pgtty,  
const char* dbName, const char* login, const char* pwd);
```

Parameters

Table 5-61 PQsetdbLogin parameters

Keyword	Description
<code>pghost</code>	Name of the host to be connected. For details, see the host column described in Connection Parameters .
<code>pgport</code>	Port number of the host server. For details, see the port column described in Connection Parameters .
<code>pgoptions</code>	Command-line options to be sent to the server during running. For details, see the options column described in Connection Parameters .

Keyword	Description
pgtty	Ignored. (This option declares the output direction of server logs.)
dbName	Name of the database to be connected. For details, see the dbname column described in Connection Parameters .
login	Username for connection. For details, see the user column described in Connection Parameters .
pwd	Password used for authentication during connection. For details, see the password column described in Connection Parameters .

Return Values

PGconn * points to the object pointer that contains a connection. The memory is allocated by the function internally.

Precautions

- This function is the predecessor of PQconnectdb with a fixed set of parameters. When an undefined parameter is called, its default value is used. Write NULL or an empty string for any one of the fixed parameters that is to be defaulted.
- If the **dbName** value contains an equal sign (=) or a valid prefix in the connection URL, it is taken as a conninfo string and passed to PQconnectdb, and the remaining parameters are consistent with PQconnectdbParams parameters.

5.6.4.1.5 PQfinish

Description

Closes the connection to the server and releases the memory used by the PGconn object.

Prototype

```
void PQfinish(PGconn* conn);
```

Parameter

Table 5-62 PQfinish parameter

Keyword	Description
conn	Object pointer that contains the connection information.

Precautions

If the server connection attempt fails (as indicated by PQstatus), the application should call PQfinish to release the memory used by the PGconn object. The PGconn pointer must not be used again after PQfinish has been called.

Example

For details, see [Typical Application Development Examples](#).

5.6.4.1.6 PQreset

Function

PQreset is used to reset the communication port to the server.

Prototype

```
void PQreset(PGconn* conn);
```

Parameter

Table 5-63 PQreset parameter

Keyword	Parameter Description
conn	The object pointer that contains the connection information.

Precautions

This function will close the connection to the server and attempt to establish a new connection to the same server by using all the parameters previously used. This function is applicable to fault recovery after a connection exception occurs.

5.6.4.1.7 PQstatus

Description

Returns the connection status.

Prototype

```
ConnStatusType PQstatus(const PGconn *conn);
```

Parameters

Table 5-64 PQstatus parameter

Keyword	Description
conn	Object pointer that contains the connection information.

Return Values

ConnStatusType indicates the connection status. The enumerated values are as follows:

```
CONNECTION_STARTED
Waiting for the connection to be established.

CONNECTION_MADE
Connection succeeded; waiting to send.

CONNECTION_AWAITING_RESPONSE
Waiting for a response from the server.

CONNECTION_AUTH_OK
Authentication received; waiting for backend startup to complete.

CONNECTION_SSL_STARTUP
Negotiating SSL encryption.

CONNECTION_SETENV
Negotiating environment-driven parameter settings.

CONNECTION_OK
Normal connection.

CONNECTION_BAD
Failed connection.
```

Precautions

The connection status can be one of the preceding values. After the asynchronous connection procedure is complete, only two of them, **CONNECTION_OK** and **CONNECTION_BAD**, can return. **CONNECTION_OK** indicates that the connection to the database is normal. **CONNECTION_BAD** indicates that the connection to the database fails. Generally, the normal state remains until PQfinish is called. However, a communication failure may cause the connection status to become to **CONNECTION_BAD** before the connection procedure is complete. In this case, the application can attempt to call PQreset to restore the communication.

Examples

For details, see [Typical Application Development Examples](#).

5.6.4.2 Database Statement Execution Functions

After the connection to the database server is successfully established, you can use the functions described in this section to execute SQL queries and commands.

5.6.4.2.1 PQexec

Description

Commits a command to the server and waits for the result.

Prototype

```
PGresult* PQexec(PGconn* conn, const char* query);
```

Parameters

Table 5-65 PQexec parameters

Keyword	Description
conn	Object pointer that contains the connection information.
query	Query string to be executed.

Return Values

PGresult indicates the object pointer that contains the query result.

Precautions

The PQresultStatus function should be called to check the return value for any errors (including the value of a null pointer, in which **PGRES_FATAL_ERROR** will be returned). The PQerrorMessage function can be called to obtain more information about such errors.

NOTICE

The command string can contain multiple SQL commands separated by semicolons (;). Multiple queries sent in a PQexec call are processed in one transaction, unless there are specific BEGIN/COMMIT commands in the query string to divide the string into multiple transactions. Note that the returned PGresult structure describes only the result of the last command executed from the string. If a command fails, the string processing stops and the returned PGresult describes the error condition.

Examples

For details, see [Typical Application Development Examples](#).

5.6.4.2.2 PQprepare

Description

Commits a request to create a prepared statement with given parameters and waits for completion.

Prototype

```
PGresult* PQprepare(PGconn* conn, const char* stmtName, const char* query, int nParams, const Oid* paramTypes);
```

Parameters

Table 5-66 PQprepare parameters

Keyword	Description
conn	Object pointer that contains the connection information.
stmtName	Name of prepare to be executed.
query	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Array of the parameter type.

Return Values

PGresult indicates the object pointer that contains the query result.

Precautions

- PQprepare creates a prepared statement for later execution with PQexecPrepared. This function allows commands to be repeatedly executed, without being parsed and planned each time they are executed. PQprepare is supported only in protocol 3.0 or later. It will fail when protocol 2.0 is used.
- This function creates a prepared statement named **stmtName** from the query string, which must contain an SQL command. **stmtName** can be "" to create an unnamed statement. In this case, any pre-existing unnamed statement will be automatically replaced. Otherwise, this is an error if the statement name has been defined in the current session. If any parameters are used, they are referred to in the query as \$1, \$2, and so on. **nParams** is the number of parameters for which types are pre-specified in the array paramTypes[]. (The array pointer can be **NULL** when **nParams** is **0**.) paramTypes[] specifies the data types to be assigned to the parameter symbols by OID. If **paramTypes** is **NULL**, or any element in the array is **0**, the server assigns a data type to the parameter symbol in the same way as it does for an untyped literal string. In addition, the query can use parameter symbols whose numbers are greater than **nParams**. Data types of these symbols will also be inferred.

NOTICE

You can also execute the **SQLPREPARE** statement to create a prepared statement that is used with PQexecPrepared. Although there is no libpq function of deleting a prepared statement, the SQL DEALLOCATE statement can be used for this purpose.

Examples

For details, see [Typical Application Development Examples](#).

5.6.4.2.3 PQclear

Description

Releases the storage associated with PGresult. Any query result should be released by PQclear when it is no longer needed.

Prototype

```
void PQclear(PGresult* res);
```

Parameter

Table 5-67 PQclear parameter

Keyword	Description
res	Object pointer that contains the query result.

Precautions

PGresult is not automatically released. That is, it does not disappear when a new query is committed or even if you close the connection. To delete it, you must call PQclear. Otherwise, memory leakage occurs.

Examples

For details, see [Typical Application Development Examples](#).

5.6.4.3 Asynchronous Command Processing Functions

The PQexec function is adequate for committing commands in common, synchronous applications. However, it has several defects, which may be important to some users:

- PQexec waits for the end of the command, but the application may have other work to do (for example, maintaining a user interface). In this case, PQexec would not want to be blocked to wait for the response.
- As the client application is suspended while waiting for the result, it is difficult for the application to determine whether to cancel the ongoing command.
- PQexec can return only one PGresult structure. If the committed command string contains multiple SQL commands, all the PGresult structures except the last PGresult are discarded by PQexec.
- PQexec always collects the entire result of the command and caches it in a PGresult. Although this mode simplifies the error handling logic for applications, it is impractical for results that contain multiple rows.

Applications that do not want to be restricted by these limitations can use the following functions that PQexec is built from: PQsendQuery and PQgetResult. The functions PQsendQueryParams, PQsendPrepare, and PQsendQueryPrepared can also be used with PQgetResult.

5.6.4.3.1 PQsendQuery

Description

Commits a command to the server without waiting for the result. If the query is successful, **1** is returned. Otherwise, **0** is returned.

Prototype

```
int PQsendQuery(PGconn* conn, const char* query);
```

Parameter

Table 5-68 PQsendQuery parameters

Keyword	Description
conn	Points to the object pointer that contains the connection information.
query	Query string to be executed.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

After PQsendQuery is successfully called, call PQgetResult one or more times to obtain the results. PQsendQuery cannot be called again (on the same connection) until PQgetResult returns a null pointer, indicating that the command execution is complete.

5.6.4.3.2 PQsendQueryParams

Function

PQsendQueryParams is used to submit a command and separate parameters to the server without waiting for the result.

Prototype

```
int PQsendQueryParams(PGconn* conn, const char* command, int nParams, const Oid* paramTypes, const char* const* paramValues, const int* paramLengths, const int* paramFormats, int resultFormat);
```

Parameter

Table 5-69 PQsendQueryParams parameters

Keyword	Parameter Description
conn	The object pointer that contains the connection information.
command	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Parameter type.
paramValues	Parameter value.
paramLengths	Parameter length.
paramFormats	Parameter format.
resultFormat	Result format.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

This function is equivalent to PQsendQuery. The only difference is that query parameters can be specified separately from the query string. The parameter processing of this function is similar to that of PQexecParams. It cannot work on connections using protocol v2.0, and it allows only one command to appear in the query string.

5.6.4.3.3 PQsendPrepare

Description

Sends a request to create a prepared statement with given parameters, without waiting for completion.

Prototype

```
int PQsendPrepare(PGconn* conn, const char* stmtName, const char* query, int nParams, const Oid* paramTypes);
```

Parameters

Table 5-70 PQsendPrepare parameters

Keyword	Parameter Description
conn	The object pointer that contains the connection information.
stmtName	Name of prepare to be executed.
query	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Array of the parameter type.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

PQsendPrepare is an asynchronous version of PQprepare. If it can dispatch a request, **1** is returned. Otherwise, **0** is returned. After a successful calling of PQsendPrepare, call PQgetResult to check whether the server successfully created the preparedStatement. PQsendPrepare parameters are handled in the same way as PQprepare parameters. Like PQprepare, PQsendPrepare cannot work on connections using protocol v2.0.

5.6.4.3.4 PQsendQueryPrepared

Description

Sends a request to execute a prepared statement with given parameters, without waiting for the result.

Prototype

```
int PQsendQueryPrepared(PGconn* conn, const char* stmtName, int nParams, const char* const* paramValues, const int* paramLengths, const int* paramFormats, int resultFormat);
```

Parameters

Table 5-71 PQsendQueryPrepared parameters

Keyword	Description
conn	Object pointer that contains the connection information.
stmtName	Name of prepare to be executed.
nParams	Parameter quantity.

Keyword	Description
paramValues	Parameter value.
paramLengths	Parameter length.
paramFormats	Parameter format.
resultFormat	Result format.

Return Values

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

This function is similar to PQsendQueryParams, but the command to be executed is specified by naming a previously-prepared statement, instead of providing a query string. The parameters of this function are processed in the same way as PQexecPrepared. Like PQexecPrepared, it cannot work on connections using protocol v2.0.

5.6.4.3.5 PQflush

Description

Flushes any queued output data to the server.

Prototype

```
int PQflush(PGconn* conn);
```

Parameters

Table 5-72 PQflush parameter

Keyword	Description
conn	Object pointer that contains the connection information.

Return Values

int indicates the operation result. If the operation is successful (or the send queue is empty), **0** is returned. If the operation fails, **-1** is returned. If all data in the send queue fails to be sent, **1** is returned. (This case occurs only when the connection is non-blocking.) The failure cause is stored in **conn->errorMessage**.

Precautions

Call PQflush after sending any command or data over a non-blocking connection. If **1** is returned, wait for the socket to become read- or write-ready. If the socket

becomes write-ready, call PQflush again. If the socket becomes read-ready, call PQconsumeInput and then call PQflush again. Repeat the operation until the value **0** is returned for PQflush. It is necessary to check read-ready and use PQconsumeInput to exhaust input because the server may prevent attempts to send data (such as NOTICE messages) to the client and does not read the client's data until the client reads its data. Once PQflush returns **0**, wait for the socket to be read-ready and then read the response as described above.

5.6.4.4 Functions for Canceling Query Processing

A client application can use the functions described in this section to cancel a command that is still being processed by the server.

5.6.4.4.1 PQgetCancel

Description

Creates a data structure that contains the information required to cancel a command issued through a specific database connection.

Prototype

```
PGcancel *PQgetCancel(PGconn *conn);
```

Parameters

Table 5-73 PQgetCancel parameter

Keyword	Description
conn	Object pointer that contains the connection information.

Return Values

PGcancel points to the object pointer that contains the cancel information.

Precautions

PQgetCancel creates a PGcancel object for a given PGconn connection object. If the given connection object (**conn**) is NULL or an invalid connection, it will return NULL. The PGcancel object is an opaque structure that cannot be directly accessed by applications. It can be transferred only to PQcancel or PQfreeCancel.

5.6.4.4.2 PQfreeCancel

Function

PQfreeCancel is used to release the data structure created by PQgetCancel.

Prototype

```
void PQfreeCancel(PGcancel* cancel);
```

Parameter

Table 5-74 PQfreeCancel parameter

Keyword	Parameter Description
cancel	Points to the object pointer that contains the cancel information.

Precautions

PQfreeCancel releases a data object previously created by PQgetCancel.

5.6.4.4.3 PQcancel

Description

Requests the server to abandon processing of the current command.

Prototype

```
int PQcancel(PGcancel *cancel, char *errbuf, int errbufsize);
```

Parameters

Table 5-75 PQcancel parameters

Keyword	Parameter Description
cancel	Points to the object pointer that contains the cancel information.
errbuf	Buffer for storing error information.
errbufsize	Size of the buffer for storing error information.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **errbuf**.

Precautions

- Successful sending does not guarantee that the request will have any effect. If the cancellation is valid, the current command is terminated early and an error is returned. If the cancellation fails (for example, because the server has processed the command), no result is returned.
- If **errbuf** is a local variable in a signal handler, you can safely call PQcancel from the signal handler. For PQcancel, the PGcancel object is read-only, so it can also be called from a thread that is separate from the thread that is operating the PGconn object.

5.6.4.5 Functions for Processing Database Result

5.6.4.5.1 PQgetvalue

Description

Returns a single threshold for a row of PGresult. Row and column numbers start from 0. The caller should not release the result directly. The result will be released when the associated PGresult handle is passed to PQclear.

Prototype

```
char* PQgetvalue(const PGresult* res, int tup_num, int field_num);
```

Parameters

Table 5-76 PQgetvalue parameters

Keyword	Description
res	Handle to the operation result.
tup_num	Number of rows.
field_num	Number of columns.

Return Values

For data in text format, PQgetvalue returns a null-terminated string representation of the threshold.

For binary data, the value is a binary representation determined by the typsend and typreceive functions of the data type.

If the threshold is empty, an empty string is returned.

5.6.4.5.2 PQnfields

Description

Returns the number of columns (fields) in each row of the query result.

Prototype

```
int PQnfields(const PGresult* res);
```

Parameters

Table 5-77 PQnfields parameters

Keyword	Description
res	Handle to the operation result.

Return Values

Value of the int type

5.6.4.5.3 PQntuples

Description

Returns the number of rows (tuples) in the query result.

Prototype

```
int PQntuples(const PGresult* res);
```

Parameters

Table 5-78 PQntuples parameters

Keyword	Description
res	Handle to the operation result.

NOTICE

PQntuples returns an integer result. An overflow may occur if the return value is out of the value range allowed in a 32-bit OS.

Return Values

Value of the int type

Examples

For details, see [Typical Application Development Examples](#).

5.6.4.5.4 PQfname

Description

Returns the column name associated with the given column number. Column numbers start from 0. The caller should not release the result directly. The result will be released when the associated PGresult handle is passed to PQclear.

Prototype

```
char* PQfname(const PGresult* res, int field_num);
```

Parameters

Table 5-79 PQfname parameters

Keyword	Description
res	Handle to the operation result.
field_num	Number of columns.

Return Values

char pointers.

5.6.4.5.5 PQresultStatus

Description

Returns the result status of a command.

Prototype

```
ExecStatusType PQresultStatus(const PGresult* res);
```

Parameter

Table 5-80 PQresultStatus parameter

Keyword	Description
res	Object pointer that contains the query result.

Return Values

PQresultStatus indicates the command execution status. The enumerated values are as follows:

PQresultStatus can return one of the following values:
PGRES_EMPTY_QUERY

The string sent to the server was empty.

PGRES_COMMAND_OK
A command that does not return data was successfully executed.

PGRES_TUPLES_OK
A query (such as SELECT or SHOW) that returns data was successfully executed.

PGRES_COPY_OUT
The data copied from the server begins to transmit.

PGRES_COPY_IN
The data copied to the server begins to transmit.

PGRES_BAD_RESPONSE
The response from the server cannot be understood.

PGRES_NONFATAL_ERROR
A non-fatal error (notification or warning) occurred.

PGRES_FATAL_ERROR
A fatal error occurred.

PGRES_COPY_BOTH
The data copied to and from the server begins to transmit. This state occurs only in streaming replication.

PGRES_SINGLE_TUPLE
PGresult contains a result tuple from the current command. This state occurs in a single-row query.

Precautions

- Note that the SELECT command that happens to retrieve zero rows still returns **PGRES_TUPLES_OK**. **PGRES_COMMAND_OK** is used for commands that can never return rows (such as INSERT or UPDATE, without return clauses). The result status **PGRES_EMPTY_QUERY** might indicate a bug in the client software.
- The result status **PGRES_NONFATAL_ERROR** will never be returned directly by PQexec or other query execution functions. Instead, such results will be passed to the notice processor.

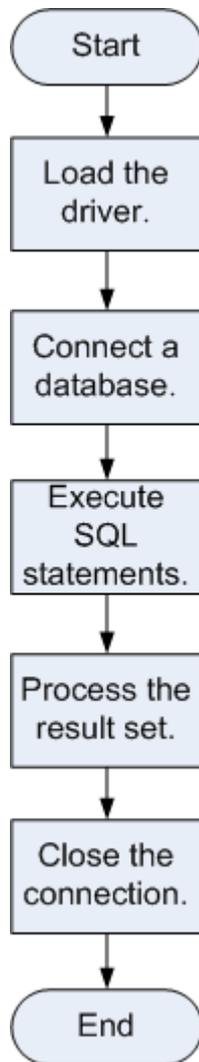
Examples

For details, see [Typical Application Development Examples](#).

5.7 Psycopg-based Development

5.7.1 Development Process

Figure 5-5 Application development process based on psycopg2



5.7.2 Development Procedure

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-81](#).

Table 5-81 Driver package download list

Version	Download Address
V2.0-3.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

- Verifying the software package integrity on a Linux server:
 - a. Upload the software package and verification package to the same directory on the VM.
 - b. Run the following command to verify the integrity of the software package:

```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

If **OK** is displayed in the command output, the verification is successful.

```
GaussDB_driver.zip: OK
```
- Verifying the software package integrity on a Windows server:
 - a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
 - b. Run the following command to obtain the hash value of the driver package:

```
certutil -hashfile {local_directory_of_the_driver_package}\{driver_package_name} sha256
```

 - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
 - Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **GaussDB_driver.zip**.Example: **certutil -hashfile C:\Users\GaussDB_driver.zip sha256**
 - c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-81**.
 - If they are consistent, the verification is successful.
 - If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Connecting to a Database

- Step 1** Prepare related drivers and dependency libraries. Obtain the package **GaussDB-Kernel_Database version number_OS information_64bit_Python.tar.gz** from the driver package.

After the decompression, the following folders are generated:

- **psycopg2**: **psycopg2** library file
- **lib**: **lib** library file

- Step 2** Load the driver.

- Before using the driver, perform the following operations:
 - a. Decompress the driver package of the corresponding version.

```
tar zxvf xxxx-Python.tar.gz
```
 - b. Copy **psycopg2** to the **site-packages** folder in the Python installation directory as the **root** user.

- ```
su root
cp psycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
```
- c. Change the **psycopg2** directory permission to **755**.  

```
chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
```
  - d. Add the **psycopg2** directory to the environment variable **\$PYTHONPATH** and validate it.  

```
export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
```
  - e. For non-database users, configure the **lib** directory in **LD\_LIBRARY\_PATH** after decompression.  

```
export LD_LIBRARY_PATH=path/to/lib:$LD_LIBRARY_PATH
```
- Load a database driver before creating a database connection.  

```
import psycopg2
```

### Step 3 Connect to a database.

Connect to the database in non-SSL mode.

1. Use the `psycopg2.connect` function to obtain the connection object.
2. Use the connection object to create a cursor object.

Connect to the database in SSL mode.

When you use `psycopy2` to connect to the GaussDB server, you can enable SSL to encrypt the communication between the client and server. To enable SSL, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

1. Use the `.ini` file (the **configparser** package of Python can parse this type of configuration file) to save the configuration information about the database connection.
2. Add SSL connection parameters **sslmode**, **sslcert**, **sslkey**, and **sslrootcert** to the connection options.
  - a. **sslmode**: For details about the options, see [Table 5-82](#).
  - b. **sslcert**: client certificate path.
  - c. **sslkey**: client key path.
  - d. **sslrootcert**: root certificate path.
3. Use the `psycopg2.connect` function to obtain the connection object.
4. Use the connection object to create a cursor object.

 **CAUTION**

To use SSL to connect to the database, ensure that the Python interpreter is compiled in the mode of generating a dynamic link library (.so) file. You can perform the following steps to check the connection mode of the Python interpreter:

1. Run the **import ssl** command in the Python interpreter to import SSL.
2. Run the **ps ux** command to query the PID of the Python interpreter. Assume that the PID is **\*\*\*\*\***.
3. In the Python interpreter CLI, run the **pmap -p \*\*\*\*\* | grep ssl** command and check whether the command output contains the path related to **libssl.so**. If it does, the Python interpreter is compiled in dynamic link mode.

**Table 5-82** sslmode options

| sslmode     | Enable SSL Encryption | Description                                                                                                                                       |
|-------------|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| disable     | No                    | SSL connection is not enabled.                                                                                                                    |
| allow       | Possible              | If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified. |
| prefer      | Possible              | If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.          |
| require     | Yes                   | SSL connection is required, but only data is encrypted. However, authenticity of the database server will not be verified.                        |
| verify-ca   | Yes                   | SSL connection is required, and the validity of the server CA must be verified.                                                                   |
| verify-full | Yes                   | The SSL connection must be enabled, which is not supported by GaussDB currently.                                                                  |

**Step 4** Run SQL statements.

1. Construct an operation statement and use %s as a placeholder. During execution, psycopg2 will replace the placeholder with the parameter value. You can add the RETURNING clause to obtain the automatically generated column values.
2. Use the cursor.execute method to execute one row of SQL statement, and use the cursor.executemany method to execute multiple rows of SQL statements.

**Step 5** Process the result set.

1. `cursor.fetchone()`: fetches the next row in a query result set and returns a sequence. If no data is available, null is returned.
2. `cursor.fetchall()`: fetches all remaining rows in a query result and returns a list. An empty list is returned when no rows are available.

**NOTE**

For database-specific data types, such as `tinyint`, the corresponding columns in the query result are character strings.

**Step 6** Close the connection.

After you complete required data operations in a database, close the database connection. Call the close method such as `connection.close()` to close the connection.

**CAUTION**

This method closes the database connection and does not automatically call `commit()`. If you close the database connection without calling `commit()`, changes will be lost.

----End

## 5.7.3 Examples: Common Operations

```
import psycopg2
import os

Obtain the username and password from environment variables.
user = os.getenv('user')
password = os.getenv('password')

Create a connection object.
conn=psycopg2.connect(database="database", user=user, password=password, host="localhost", port=port)
cur=conn.cursor() # Create a pointer object.

Create a connection object (using SSL).
conn = psycopg2.connect(dbname="database", user=user, password=password, host="localhost", port=port,
 sslmode="verify-ca", sslcert="client.crt", sslkey="client.key", sslrootcert="cacert.pem")
Note: If sslcert, sslkey, and sslrootcert are not set, the following files in the .postgresql directory of the
current user are used by default: client.crt, client.key, and root.crt.

Create a table.
cur.execute("CREATE TABLE student(id integer,name varchar,sex varchar);")

Insert data.
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(1,'Aspirin','M'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(2,'Taxol','F'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(3,'Dixheral','M'))

Insert data in batches.
stus = ((4,'John','M'),(5,'Alice','F'),(6,'Peter','M'))
cur.executemany("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",stus)

Obtain the result.
cur.execute("SELECT * FROM student")
results=cur.fetchall()
print (results)
```

```
Perform a commit.
conn.commit()

Insert a data record.
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(7,'Lucy','F'))

Perform a rollback.
conn.rollback()

Close the connection.
cur.close()
conn.close()

Common connection modes of psycopg2
1. conn = psycopg2.connect(dbname="dbname", user=user, password=password, host="localhost",
port=port)
2. conn = psycopg2.connect(f"dbname=dbname user={user} password={password} host=localhost
port=port")
3. Using logs
import logging
import psycopg2
from psycopg2.extras import LoggingConnection
import os

Obtain the username and password from environment variables.
user = os.getenv('user')
password = os.getenv('password')

logging.basicConfig(level=logging.DEBUG) # Log level
logger = logging.getLogger(__name__)

db_settings = {
 "user": user,
 "password": password,
 "host": "localhost",
 "database": "dbname",
 "port": port
}

LoggingConnection records all SQL statements by default. You can filter unnecessary or sensitive SQL
statements. Example of filtering password-related SQL statements:
class SelfLoggingConnection(LoggingConnection):

 def filter(self, msg, curs):
 if db_settings['password'] in msg.decode():
 return b'queries containing the password will not be recorded'
 return msg

conn = psycopg2.connect(connection_factory=SelfLoggingConnection, **db_settings)
conn.initialize(logger)
```

#### NOTE

- By default, **LoggingConnection** records all SQL information and does not anonymize sensitive information. You can use the filter function to define the output SQL content.
- The log function is an additional function provided by psycopg2 for developers to explicitly debug full SQL statements. By default, the log function is not used. This function prints SQL statements before psycopg2 executes SQL statements. However, the SQL statements can be printed only when the log level is **DEBUG**. This function is not a default function. It is used only when there are special requirements. You are advised not to use this function unless there are special requirements. For details, visit <https://www.psycopg.org/docs/extras.html?highlight=loggingconnection>.

## 5.7.4 Psycopg API Reference

Psycopg APIs are a set of methods provided for users. This section describes some common APIs.

### 5.7.4.1 psycopg2.connect()

#### Description

This method creates a database session and returns a new connection object.

#### Prototype

```
import os
conn=psycopg2.connect(dbname="test",user=os.getenv('user'),password=os.getenv('password'),host="127.0.0.1",port="5432")
```

#### Parameters

**Table 5-83** psycopg2.connect parameters

| Keyword                   | Description                                                                                                                                                                                                                                                              |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dbname                    | Database name.                                                                                                                                                                                                                                                           |
| user                      | Username.                                                                                                                                                                                                                                                                |
| password                  | Password.                                                                                                                                                                                                                                                                |
| host                      | IP address of the database. You can specify multiple IP addresses and separate them with commas (.). By default, the UDS is used.                                                                                                                                        |
| port                      | Connection port number. The default value is <b>5432</b> . If the host has multiple IP addresses and the port numbers are the same, specify one port number. Otherwise, the port numbers must correspond to the IP addresses one by one and are separated by commas (.). |
| sslmode                   | SSL mode, which is used for SSL connection.                                                                                                                                                                                                                              |
| sslcert                   | Path of the client certificate, which is used for SSL connection.                                                                                                                                                                                                        |
| sslkey                    | Path of the client key, which is used for SSL connection.                                                                                                                                                                                                                |
| sslrootcert               | Path of the root certificate, which is used for SSL connection.                                                                                                                                                                                                          |
| hostaddr                  | IP address of the database.                                                                                                                                                                                                                                              |
| connect_timeout           | Client connection timeout interval.                                                                                                                                                                                                                                      |
| client_encoding           | Encoding format of the client.                                                                                                                                                                                                                                           |
| application_name          | Value of <b>application_name</b> .                                                                                                                                                                                                                                       |
| fallback_application_name | Rollback value of <b>application_name</b> .                                                                                                                                                                                                                              |

| Keyword              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| keepalives           | Determines whether to enable the TCP connection on the client. The default value is <b>1</b> , indicating that the TCP connection is enabled. The value <b>0</b> indicates that the TCP connection is disabled. If the UDS connection is used, ignore this parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| options              | Specifies the command line options sent to the server when the connection starts.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| keepalives_idle      | Describes inactivity before keepalive messages are sent to the server. If <b>keepalive</b> is disabled, ignore this parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| keepalives_interval  | Determines whether keepalive messages that are not confirmed by the server need to be resent. If <b>keepalive</b> is disabled, ignore this parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| keepalives_count     | Specifies the number of TCP connections that may be lost before the client is disconnected from the server.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| replication          | Ensures that the connection uses the replication protocol instead of the common protocol.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| requiressl           | Supports the SSL mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| sslcompression       | Specifies the SSL compression. If this parameter is set to <b>1</b> , the data sent through the SSL connection is compressed. If this parameter is set to <b>0</b> , the compression is disabled. If no SSL connection is established, ignore this parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| sslcrll              | Specifies the path of the certificate revocation list (CRL), which is used to check whether the SSL server certificate is available.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| requirepeer          | Specifies the OS username of the server.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| target_session_attrs | Specifies the type of the host to be connected. The connection is successful only when the host type is the same as the configured value. This parameter is verified only when multiple IP addresses are specified. The rules for setting <b>target_session_attrs</b> are as follows: <ul style="list-style-type: none"><li>• <b>any</b>: All types of hosts can be connected.</li><li>• <b>read-write</b>: The connection is set up only when the connected host is readable and writable.</li><li>• <b>read-only</b>: Only readable hosts can be connected.</li><li>• <b>primary</b> (default value): Only the primary node in the primary/standby system can be connected.</li><li>• <b>standby</b>: Only the standby node in the primary/standby system can be connected.</li><li>• <b>prefer-standby</b>: The system first attempts to find a standby node for connection. If all hosts in the <b>hosts</b> list fail to be connected, try the <b>any</b> mode.</li></ul> |

| Keyword          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tcp_user_timeout | Specifies the maximum duration for which transmitted data can remain unacknowledged before the TCP connection is forcibly closed on an OS that supports the <b>TCP_USER_TIMEOUT</b> socket option. The value <b>0</b> indicates that the default value is used. Ignore this parameter for UDS connections.                                                                                                                                                                |
| rw_timeout       | Sets the read and write timeout interval of the client connection.<br><br>When the timeout is triggered on the libpq and the connection is closed, the running services delivered by the libpq to the database are forcibly terminated. This capability is controlled by the GUC parameter <b>check_disconnect_query</b> . If this parameter is set to <b>on</b> , the capability is supported. If this parameter is set to <b>off</b> , the capability is not supported. |

## Return Values

Connection object (for connecting to a database instance)

## Example

For details, see [Examples: Common Operations](#).

### 5.7.4.2 connection.cursor()

## Function

This method returns a new cursor object.

## Prototype

```
cursor(name=None, cursor_factory=None, scrollable=None, withhold=False)
```

## Parameter

**Table 5-84** connection.cursor parameters

| Keyword        | Description                                                       |
|----------------|-------------------------------------------------------------------|
| name           | Cursor name. The default value is <b>None</b> .                   |
| cursor_factory | Creates a non-standard cursor. The default value is <b>None</b> . |
| scrollable     | Sets the SCROLL option. The default value is <b>None</b> .        |
| withhold       | Sets the HOLD option. The default value is <b>False</b> .         |

## Return Value

Cursor object (used for cursors that are programmed using Python in the entire database)

## Example

For details, see [Examples: Common Operations](#).

### 5.7.4.3 cursor.execute(query,vars\_list)

## Description

This method executes the parameterized SQL statements (that is, placeholders instead of SQL literals). The psycopg2 module supports placeholders marked with %s.

## Prototype

```
cursor.execute(query,vars_list)
```

## Parameters

Table 5-85 cursor.execute

| Keyword   | Description                                                   |
|-----------|---------------------------------------------------------------|
| query     | SQL statement to be executed.                                 |
| vars_list | Variable list, which matches the %s placeholder in the query. |

## Return Value

None

## Examples

For details, see [Examples: Common Operations](#).

### 5.7.4.4 cursor.executemany(query,vars\_list)

## Description

This method executes an SQL command against all parameter sequences or mappings found in the sequence SQL.

## Prototype

```
cursor.executemany(query,vars_list)
```

## Parameters

**Table 5-86** cursor.executemany

| Keyword   | Description                                                   |
|-----------|---------------------------------------------------------------|
| query     | SQL statement to be executed.                                 |
| vars_list | Variable list, which matches the %s placeholder in the query. |

## Return Value

None

## Examples

For details, see [Examples: Common Operations](#).

### 5.7.4.5 connection.commit()

#### Function

This method commits the currently pending transaction to the database.

---

 **CAUTION**

By default, Psycopg opens a transaction before executing the first command. If **commit()** is not called, the effect of any data operation will be lost.

---

#### Prototype

```
connection.commit()
```

#### Parameter

None

#### Return Value

None

#### Example

For details, see [Examples: Common Operations](#).

### 5.7.4.6 connection.rollback()

#### Function

This method rolls back the current pending transaction.

---

 **CAUTION**

If you close the connection using **close()** but do not commit the change using **commit()**, an implicit rollback will be performed.

---

## Prototype

```
connection.rollback()
```

## Parameter

None

## Return Value

None

## Example

For details, see [Examples: Common Operations](#).

### 5.7.4.7 cursor.fetchone()

## Function

This method extracts the next row of the query result set and returns a tuple.

## Prototype

```
cursor.fetchone()
```

## Parameter

None

## Return Value

A single tuple is the first result in the result set. If no more data is available, **None** is returned.

## Example

For details, see [Examples: Common Operations](#).

### 5.7.4.8 cursor.fetchall()

## Function

This method gets all the (remaining) rows of the query result and returns them as a list of tuples.

## Prototype

```
cursor.fetchall()
```

## Parameter

None

## Return Value

Tuple list, which contains all results of the result set. An empty list is returned when no rows are available.

## Example

For details, see [Examples: Common Operations](#).

### 5.7.4.9 cursor.close()

## Function

This method closes the cursor of the current connection.

## Prototype

```
cursor.close()
```

## Parameter

None

## Return Value

None

## Example

For details, see [Examples: Common Operations](#).

### 5.7.4.10 connection.close()

## Function

This method closes the database connection.

---

**CAUTION**

This method closes the database connection and does not automatically call **commit()**. If you just close the database connection without calling **commit()** first, changes will be lost.

---

## Prototype

```
connection.close()
```

## Parameter

None

## Return Value

None

## Example

For details, see [Examples: Common Operations](#).

# 5.8 ECPG-based Development

Embedded SQL C Preprocessor (ECPG) for GaussDB Kernel is an embedded SQL preprocessor for C programs. An embedded SQL program consists of code written in an ordinary programming language, in this case C, mixed with SQL commands in specially marked sections. To build the program, the source code (\*.pgc) is first passed through the embedded SQL preprocessor, which converts it to an ordinary C program (\*.c), and afterwards it can be processed by a C compiler. Converted ECPG applications call functions in the libpq library through the embedded SQL library (ecpglib), and communicate with the GaussDB Kernel server using the normal frontend-backend protocol.

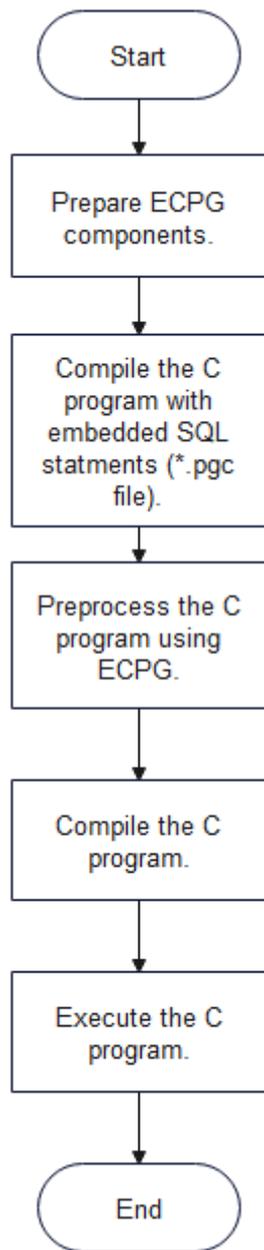
Programs written for the embedded SQL interface are normal C programs with special code inserted to perform database-related actions. This special code always has the form:

```
EXEC SQL ...;
```

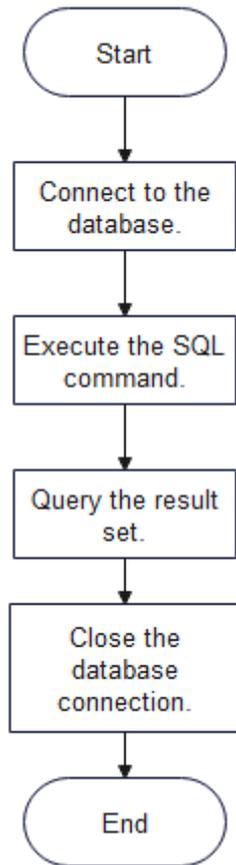
These statements syntactically take the place of a C statement. Depending on the particular statement, they can appear at the global level or within a function. Embedded SQL statements follow the case-sensitivity rules of normal SQL code, and allow nested C code-style comments (part of the SQL standard). However, the C part of the program follows the standards of the C program and does not support nested comments.

## 5.8.1 Development Process

Figure 5-6 ECPG-based development process



**Figure 5-7** Embedded SQL-C program development process



## 5.8.2 ecpg Components

- Platforms supported by ecpg

**Table 5-87** Platforms supported by ecpg

| OS              | Platform |
|-----------------|----------|
| EulerOS V2.0SP5 | x86_64   |
| EulerOS V2.0SP9 | Arm64    |

- ecpg components
  - ecpg: an executable binary file used to preprocess C programs with embedded SQL statements.
  - libecpg: dynamic library provided by ecpg to implement connections, SQL statements, and transactions, including **libecpg.so**, **libecpg.so.6**, and **libecpg.so.6.4**. It is referenced by the **-lecpg** parameter during C program compilation and execution.
  - libpgtypes: dynamic library provided by ecpg for operating data of the numeric, date, timestamp, and interval types, including **libpgtypes.so**, **libecpg.so.6**, and **libecpg.so.6.4**. The library is referenced by the **-lpgtypes** parameter during C program compilation and execution.

- Paths for obtaining ecpg components
  - ecpg binary file: `$GAUSSHOME/bin`
  - Dynamic library on which ecpg depends: `$GAUSSHOME/lib`
  - Header file required by ecpg: `$GAUSSHOME/include/ecpg`

### 5.8.3 ecpg Preprocessing and Compiling

Prepare C programs with embedded SQL statements with the extension .pgc. ecpg converts them into C programs that can be compiled by the C compiler.

The generated C program is compiled into an executable file by the GCC compiler. The executable file is run to enable the client program to access the database. For details, see [Examples](#).

- ecpg preprocessing and C compilation process
  - a. Preprocessing: `ecpg -I $GAUSSHOME/include -o test.c test.pgc`  
To execute ecpg preprocessing, run the following command:  
`ecpg [OPTION]...`  
The options are as follows:
    - **-o OUTFILE**: writes the result to OUTFILE, which is a C file.
    - **-I DIRECTORY**: path of the header file.
    - **-c**: automatically generates a C file.
    - **--version**: checks the current ecpg version.
  - b. Compilation: `gcc -I $GAUSSHOME/include/ecpg -I $GAUSSHOME/include -I $GAUSSHOME/include/postgresql/server/ -L $GAUSSHOME/lib -lecpg -lrt -lpq -lpqtypes -lpthread test_ecpg.c -o test_ecpg`
  - c. Execution: `./test`

#### NOTICE

- ecpg is a compilation preprocessing tool. If an error message is displayed indicating that the related header file or function implementation cannot be found during preprocessing or compilation, you can specify the header file or link the dynamic library as required.
- ecpg requires compilation preprocessing tools such as GCC and ld. You are advised to use GCC 7.3.0.
- Other dynamic libraries and header files on which ecpg depends are usually stored in `$GAUSSHOME/include/libpq` and `$GAUSSHOME/include`.
- Common dynamic library dependencies during compilation include: `-lpq`, `-lpq_ce`, and `-lpthread`. If the libpq communications library is required during development, connect to `-lpq` and `-lpq_ce`. If the multi-thread connection is required during development, connect to `-lpthread`.

## 5.8.4 Managing Database Connections

This section describes how to establish and switch a database connection.

### 5.8.4.1 Connecting to a Database

Connect to a database.

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

The target can be declared using the following methods. The italic part is a variable. Change it based on the actual situation.

- *dbname[@hostname][:port]*
- *tcp:postgresql://hostname[:port][/dbname][?options]*
- *unix:postgresql://hostname[:port][/dbname][?options]*
- An SQL string containing one of the above forms

There are also different ways to specify the username:

- *username/password*
- *username SQLIDENTIFIED BY password*
- *username USING password*

As mentioned above, the **username** and **password** parameters can be an SQL identifier, an SQL string, or a reference to a character variable.

**connection\_name** indicates the connection name. If a program uses only one connection, you can omit it. The most recently opened connection becomes the current connection.

Example:

```
#include <stdlib.h>
EXEC SQL CONNECT TO mydb@sql.mydomain.com;

EXEC SQL CONNECT TO unix:postgresql://sql.mydomain.com/mydb AS myconnection USER username;

EXEC SQL BEGIN DECLARE SECTION;
/* The values of target, user, and passwd must be read from environment variables or configuration files.
Environment variables need to be configured as required. If no environment variable is used, a character
string can be directly assigned. */
const char *target = getenv("EXAMPLE_TARGET_ENV");
const char *user = getenv("EXAMPLE_USERNAME_ENV");
const char *passwd = getenv("EXAMPLE_PASSWD_ENV");
EXEC SQL END DECLARE SECTION;
...
EXEC SQL CONNECT TO :target USER :user USING :passwd;
/* or EXEC SQL CONNECT TO :target USER :user/:passwd; */
```

For details about the complete example, see the connection syntax example in [CONNECT](#).

#### NOTE

- In the last form, character variables are referenced. For details about how to reference C variables in SQL statements, see [Host Variables](#).
- The format of the connection target is not described in the SQL standard. Therefore, to develop a portable application, you can use the method in the last example to encapsulate the connection target string into a variable.

**NOTICE**

- If **ip-port** is specified in the connection statement, username and password must be specified. This rule is determined by the GaussDB Kernel kernel communication authentication. If **ip-port** is not specified, the local *\$PGPORT* (UDS) is used for communication.
- If the SSL protocol is used for connection, run the **tcp:postgresql://hostname[:port]/[dbname][?options]** command and set **sslmode** to **disable** \require in **options**.

### 5.8.4.2 Managing Connections

SQL statements in embedded SQL programs are by default executed on the current connection, that is, the most recently opened one. If an application needs to manage multiple connections, use either of the following methods:

- **Method 1: Explicitly select a connection for each SQL statement.**  
EXEC SQL AT connection-name SELECT ...;  
This method is particularly suitable if the application needs to use several connections in mixed order.  
If the application uses multiple threads of execution, they cannot share a connection concurrently. You must either explicitly control access to the connection (using mutexes) or use a connection for each thread.
- **Method 2: Execute a statement to switch the connection.**  
EXEC SQL SET CONNECTION connection-name;  
This method is particularly suitable if many statements are executed on the same connection.

An example of managing connections is as follows:

```
#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;
char dbname[1024];
EXEC SQL END DECLARE SECTION;

int main()
{
 EXEC SQL CONNECT TO testdb1 AS con1 USER testuser;
 EXEC SQL CONNECT TO testdb2 AS con2 USER testuser;
 EXEC SQL CONNECT TO testdb3 AS con3 USER testuser;

 /* This query will be executed in the most recently opened database testdb3. */
 EXEC SQL SELECT current_database() INTO :dbname;
 printf("current=%s (should be testdb3)\n", dbname);

 /* Use AT to run a query in testdb2. */
 EXEC SQL AT con2 SELECT current_database() INTO :dbname;
 printf("current=%s (should be testdb2)\n", dbname);

 /* Switch to connection to testdb1. */
 EXEC SQL SET CONNECTION con1;

 EXEC SQL SELECT current_database() INTO :dbname;
 printf("current=%s (should be testdb1)\n", dbname);

 EXEC SQL DISCONNECT ALL;
 return 0;
}
```

Example output:

```
current=testdb3 (should be testdb3)
current=testdb2 (should be testdb2)
current=testdb1 (should be testdb1)
```

#### NOTICE

- In multi-thread mode, different threads cannot use the same connection name. The connection name of each thread must be unique.
- A connection must be established and closed in the same process or thread.

## 5.8.5 Running SQL Commands

The format of embedded SQL commands is EXEC SQL [Command]. In embedded SQL applications, you can run common standard SQL statements supported by GaussDB Kernel or extended SQL statements provided by ECPG.

### 5.8.5.1 Running SQL Statements

**Step 1** Create a table.

```
EXEC SQL CREATE TABLE foo (a int, b varchar);
```

**Step 2** Insert a row.

```
EXEC SQL INSERT INTO foo VALUES (5, 'abc');
```

**Step 3** Delete a row.

```
EXEC SQL DELETE FROM foo WHERE a = 5;
```

**Step 4** Update table data.

```
EXEC SQL UPDATE foo SET b = 'gdp' WHERE a = 7;
```

**Step 5** Query data in a single row.

```
EXEC SQL SELECT a INTO :var_a FROM foo WHERE b = 'def';
```

----End

A complete example is as follows:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main ()
{
 ECPGdebug (1, stderr);

 EXEC SQL BEGIN DECLARE SECTION;
 int var_a;
 EXEC SQL END DECLARE SECTION;

 EXEC SQL CONNECT TO postgres;
 // Create a table.
 EXEC SQL CREATE TABLE foo (a int, b varchar);
 // Insert data.
 EXEC SQL INSERT INTO foo VALUES (5, 'abc');
 EXEC SQL INSERT INTO foo VALUES (6, 'def');
 EXEC SQL INSERT INTO foo VALUES (7, 'ghi');

 // Delete a row.
 EXEC SQL DELETE FROM foo WHERE a = 5;
 // Update table data.
```

```
EXEC SQL UPDATE foo SET b = 'gdp' WHERE a = 7;
// Query table data in a single row.
EXEC SQL SELECT a INTO :var_a FROM foo WHERE b = 'def';
// Print the query results.
printf("select res is %d\n", var_a);

EXEC SQL DISCONNECT;

return 0;
}
```

### 5.8.5.2 Using Cursors

To retrieve a result set holding multiple rows, an application has to declare a cursor and fetch each row from the cursor.

**Step 1** Declare a cursor.

```
EXEC SQL DECLARE c CURSOR FOR select * from tb1;
```

**Step 2** Open a cursor.

```
EXEC SQL OPEN c;
```

**Step 3** Fetch a row of data from a cursor.

```
EXEC SQL FETCH 1 in c into :a, :str;
```

**Step 4** Close a cursor.

```
EXEC SQL CLOSE c;
```

----End

For details about how to use cursors, see [DECLARE](#). For details about the [FETCH](#) command, see [FETCH](#).

A complete example is as follows:

```
#include <string.h>
#include <stdlib.h>

int main(void)
{
 exec sql begin declare section;
 int *a = NULL;
 char *str = NULL;
 exec sql end declare section;

 int count = 0;
 exec sql connect to postgres ;
 exec sql set autocommit to off;
 exec sql begin;
 exec sql drop table if exists tb1;
 exec sql create table tb1(id int, info text);
 exec sql insert into tb1 (id, info) select generate_series(1, 100000), 'test';
 exec sql select count(*) into :a from tb1;
 printf ("a is %d\n", *a);
 exec sql commit;

 // Define a cursor.
 exec sql declare c cursor for select * from tb1;
 // Open the cursor.
 exec sql open c;
 exec sql whenever not found do break;
 while(1) {
 // Capture data.
 exec sql fetch 1 in c into :a, :str;
 count++;
 if (count == 100000) {
 printf("Fetch res: a is %d, str is %s", *a, str);
 }
 }
}
```

```
 }
 }
 // Close the cursor.
 exec sql close c;
 exec sql set autocommit to on;
 exec sql drop table tb1;
 exec sql disconnect;

 ECPGfree_auto_mem();
 return 0;
}
```

### 5.8.5.3 Transaction

In the default mode, statements are committed only when EXEC SQL COMMIT is issued. The embedded SQL interface also supports autocommit of transactions by executing the **EXEC SQL SET AUTOCOMMIT TO ON** statement. In autocommit mode, each command is automatically committed unless it is inside an explicit transaction block. This mode can be explicitly turned off by using **EXEC SQL SET AUTOCOMMIT TO OFF**.

Common transaction management commands are as follows:

- **EXEC SQL COMMIT**: commits an ongoing transaction.
- **EXEC SQL ROLLBACK**: rolls back an ongoing transaction.
- **EXEC SQL SET AUTOCOMMIT TO ON**: enables the autocommit mode.
- **SET AUTOCOMMIT TO OFF**: disables the autocommit mode. This is the default mode.

### 5.8.5.4 Prepared Statements

Prepared statements can be used when the value passed to an SQL statement is unknown at compile time or the same statement will be used multiple times.

- Statements are prepared using the **PREPARE** command. For the values that are not known yet, use the question mark (?) as the placeholder.  
EXEC SQL PREPARE stmt1 FROM "SELECT oid, datname FROM pg\_database WHERE oid = ?";
- If a statement returns a single row, the application can call EXECUTE after PREPARE to execute the statement, supplying the actual values for the placeholders with a USING clause:  
EXEC SQL EXECUTE stmt1 INTO :dboid, :dbname USING 1;
- If a statement returns multiple rows, the application can use a cursor declared based on the prepared statement. To bind input parameters, the cursor must be opened with a USING clause:  
EXEC SQL PREPARE stmt1 FROM "SELECT oid, datname FROM pg\_database WHERE oid > ?";  
EXEC SQL DECLARE foo\_bar CURSOR FOR stmt1;  
/\* When the end of the result set is reached, exit the while loop. \*/  
EXEC SQL WHENEVER NOT FOUND DO BREAK;  
EXEC SQL OPEN foo\_bar USING 100;  
...  
while (1)  
{  
 EXEC SQL FETCH NEXT FROM foo\_bar INTO :dboid, :dbname;  
 ...  
}  
EXEC SQL CLOSE foo\_bar;
- When a prepared statement is no longer needed, it should be deallocated.  
EXEC SQL DEALLOCATE PREPARE name;

## 5.8.5.5 Embedded SQL Commands

### 5.8.5.5.1 ALLOCATE DESCRIPTOR

#### Function

Allocates a newly named SQL descriptor area.

#### Syntax

```
ALLOCATE DESCRIPTOR name
```

#### Parameter Description

**name**

SQL descriptor name. It is case sensitive and is an SQL identifier or a host variable.

#### Example

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
```

#### Helpful Links

[DEALLOCATE DESCRIPTOR](#), [GET DESCRIPTOR](#), and [SET DESCRIPTOR](#)

### 5.8.5.5.2 CONNECT

#### Description

Establishes a connection between the client and the SQL server.

#### Syntax

```
CONNECT TO connection_target [AS connection_name] [USER connection_user]
```

#### Parameters

- **connection\_target**  
Specifies the target server to be connected in one of the following formats:
  - [ *database\_name* ] [ @ *host* ] [ : *port* ]: connection over TCP/IP.
  - unix: postgresql://*host* [ : *port* ] / [ *database\_name* ] [ ? *connection\_option* ]: connection over UDS.
  - tcp: postgresql://*host* [ : *port* ] / [ *database\_name* ] [ ? *connection\_option* ]: connection over TCP/IP.
  - **SQL string constant**: one of the preceding forms.
- **connection\_name**  
An optional identifier used for the connection, which can be referenced in other commands. It can be an SQL identifier or a host variable.
- **connection\_user**  
Username for database connection.

You can use *user\_name/password*, *user\_name* **SQLIDENTIFIED BY** *password*, or *user\_name* **USING** *password* to specify the username and password.

The username and password can be SQL identifiers, string constants, or host variables.

#### NOTE

In the preceding parameters, the information in italics refers to variables. Replace them based on the actual situation.

## Examples

Here are several variants of specifying connection parameters:

```
EXEC SQL CONNECT TO "connectdb" AS main;
EXEC SQL CONNECT TO "connectdb" AS second;
EXEC SQL CONNECT TO 'connectdb' AS main;
EXEC SQL CONNECT TO REGRESSDB1 as main;
EXEC SQL CONNECT TO connectdb AS :id;
EXEC SQL CONNECT TO connectdb AS main USER connectuser/connectdb;
EXEC SQL CONNECT TO connectdb AS main USER connectuser USING "connectdb";
EXEC SQL CONNECT TO connectdb AS main;
EXEC SQL CONNECT TO tcp:postgresql://localhost/connectdb USER connectuser IDENTIFIED BY *****;
EXEC SQL CONNECT TO tcp:postgresql://localhost:$PORT/connectdb USER connectuser SQLIDENTIFIED BY
*****;
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb USER connectuser SQLIDENTIFIED BY
"*****";
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb USER connectuser USING "*****";
```

Example of the connection syntax:

```
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
 // Define the host by defining columns such as database and password required by the connection string.
 // The actual values are read from environment variables or configuration files. Environment variables need to
 // be configured as required. If no environment variable is used, a character string can be directly assigned.
 exec sql begin declare section;
 const int max_str_len = 200;
 char db[max_str_len] = getenv("EXAMPLE_DATABASENAME_ENV");
 char pw[max_str_len] = getenv("EXAMPLE_PASSWD_ENV");
 char new_pw[max_str_len] = getenv("EXAMPLE_NEW_PASSWD_ENV");
 exec sql end declare section;

 // Print debug logs.
 ECPGdebug(1, stderr);

 // The connection statement involves the database, username, and password. The user must be created
 // in advance and have related operation permissions.

 // Connection mode: EXEC SQL CONNECT TO [database_name] [@host] [:port] [USER
 connection_user]
 // Case 1: Use the default local connection mode to connect to the postgres database.
 exec sql connect to postgres;
 // Case 2: Use the default local connection mode to connect to the postgres database. The connection
 // alias is conn1.
 exec sql connect to postgres as conn1;
 // Case 3: Use the ip+port mode (localhost indicates the local IP address listened by the database, and
 // $PORT indicates the listening port of the database) to connect to the connectdb database, specify the
 // database alias, and specify the user password.
 exec sql connect to connectdb@localhost:$PORT as conn2 user connectuser using :pw;
 // Case 4: Use the ip+port mode (127.0.0.1 indicates the local IP address listened by the database, and
 // $PORT indicates the listening port of the database) to connect to the connectdb database, specify the
 // database alias, and specify the user password.
```

```
exec sql connect to connectdb@127.0.0.1:$PORT as conn3 user connectuser sqlidentified by :pw;
// Case 5: Close the connection to the database.
exec sql disconnect postgres;
exec sql disconnect conn1;
exec sql disconnect conn2;
exec sql disconnect conn3;

// Connection mode: EXEC SQL CONNECT TO <tcp|unix>:<gaussdb|postgres>://host
[:port]/[database_name][?connection_option]
// Case 1: Replace the URL variables with the host variables pw and db.
strcpy(pw, new_pw);
strcpy(db, "tcp:postgres://localhost/connectdb");
exec sql connect to :db user connectuser using :pw;
// Case 2: 127.0.0.1 indicates the IP address listened by the database, and connectdb indicates the
database.
exec sql connect to tcp:postgres://127.0.0.1/connectdb as conn4 user connectuser using :pw;
// Case 3: 127.0.0.1 indicates the IP address listened by the database, connectdb indicates the database,
and connect_timeout=14 indicates the connection string configuration parameter.
exec sql connect to tcp:gaussdb://localhost/connectdb?connect_timeout=14 as conn5 user connectuser
sqlidentified by :pw;
// Case 4: Close all connections.
exec sql close all;

// Connect to the database and execute the service.
exec sql connect to tcp:postgres://127.0.0.1/connectdb as conn4 user connectuser using :pw;
exec sql set autocommit = on;
exec sql create table t1(a int);
exec sql insert into t1 values(1),(2);
exec sql select a from t1 where a > 1;
exec sql drop table t1;
exec sql disconnect current;
return 0;
}
```

#### Example of using a host variable to specify connection parameters:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
/* The values of dbname, user, and pwd must be read from environment variables or configuration files.
Environment variables need to be configured as required. If no environment variable is used, a character
string can be directly assigned. */
char *dbname = getenv("EXAMPLE_DBNAME_ENV"); /* Database name */
char *user = getenv("EXAMPLE_USERNAME_ENV"); /* Username for connection */
char *pwd = getenv("EXAMPLE_PASSWD_ENV"); /* Password */
char *connection = "tcp:postgres://localhost:$PORT/testdb"; /* Connection string */
char ver[256]; /* Buffer for storing version strings */
EXEC SQL END DECLARE SECTION;

ECPGdebug(1, stderr);
EXEC SQL CONNECT TO :dbname;
EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
EXEC SQL SELECT version() INTO :ver;
EXEC SQL DISCONNECT;

printf("version: %s\n", ver);
EXEC SQL CONNECT TO :connection USER :user USING :pwd;
EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
EXEC SQL SELECT version() INTO :ver;
EXEC SQL DISCONNECT;

printf("version: %s\n", ver);
return 0;
}
```

## Helpful Links

[DISCONNECT](#) and [SET CONNECTION](#)

### 5.8.5.5.3 DEALLOCATE DESCRIPTOR

#### Function

Deallocates a SQL descriptor area.

#### Syntax

```
DEALLOCATE DESCRIPTOR name
```

#### Parameter Description

**name**

SQL descriptor name. It is case sensitive and is an SQL identifier or a host variable.

#### Example

```
DEALLOCATE DESCRIPTOR mydesc;
```

## Helpful Links

[ALLOCATE DESCRIPTOR](#), [GET DESCRIPTOR](#), and [SET DESCRIPTOR](#)

### 5.8.5.5.4 DECLARE

#### Description

Declares a cursor for iterating over the result set of a prepared statement. This command is slightly semantically different from the SQL command **DECLARE**: Whereas the latter executes a query and prepares the result set for retrieval, this embedded SQL command merely declares a name as a "loop variable" for iterating over the result set of a query; the actual execution happens when the cursor is opened with the **OPEN** command.

#### Syntax

```
DECLARE cursor_name [BINARY] [NO SCROLL] CURSOR [{ WITH | WITHOUT } HOLD] FOR
prepared_name
DECLARE cursor_name [BINARY] [NO SCROLL] CURSOR [{ WITH | WITHOUT } HOLD] FOR query
```

#### Parameters

- **cursor\_name**  
Cursor name, which is case sensitive. It can be an SQL identifier or a host variable.
- **prepared\_name**  
Name of the prepared query, which can be an SQL identifier or a host variable.
- **query**

A SELECT command for providing the rows to be returned by the cursor.

 **NOTE**

For details about the cursor options, see [DECLARE](#).

## Examples

Examples of declaring a cursor used for query:

```
EXEC SQL DECLARE C CURSOR FOR SELECT * FROM My_Table;
EXEC SQL DECLARE C CURSOR FOR SELECT Item1 FROM T;
EXEC SQL DECLARE cur1 CURSOR FOR SELECT version();
```

Example of declaring a cursor for a prepared statement:

```
EXEC SQL PREPARE stmt1 AS SELECT version();
EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
```

## Helpful Links

[OPEN](#)

### 5.8.5.5.5 DESCRIBE

## Function

Retrieves metadata information for the result columns contained in prepared statements.

## Syntax

```
DESCRIBE [OUTPUT] prepared_name USING SQL DESCRIPTOR descriptor_name
DESCRIBE [OUTPUT] prepared_name INTO SQL DESCRIPTOR descriptor_name
DESCRIBE [OUTPUT] prepared_name INTO sqlda_name
```

## Parameter Description

- **prepared\_name**  
Name of a prepared statement, which can be an SQL identifier or a host variable.
- **descriptor\_name**  
Descriptor name, which is case sensitive. It can be an SQL identifier or a host variable.
- **sqlda\_name**  
Name of an SQLDA variable.

## Example

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
EXEC SQL PREPARE stmt1 FROM :sql_stmt;
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :charvar = NAME;
EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
```

## Helpful Links

[ALLOCATE DESCRIPTOR](#) and [GET DESCRIPTOR](#)

### 5.8.5.5.6 DISCONNECT

#### Function

Closes one or all database connections.

#### Syntax

```
DISCONNECT connection_name
DISCONNECT [CURRENT]
DISCONNECT DEFAULT
DISCONNECT ALL
```

#### Parameter Description

- **connection\_name**  
Name of the database connection established by the CONNECT command.
- **current**  
Closes the current connection, which can be a recently opened connection or a connection set by the SET CONNECTION command. This is also the default if no parameter is passed to the DISCONNECT command.
- **default**  
Closes the default connection.
- **all**  
Closes all open connections.

#### Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
 /* Create the testdb database in advance. */
 EXEC SQL CONNECT TO testdb AS DEFAULT USER testuser;
 EXEC SQL CONNECT TO testdb AS con1 USER testuser;
 EXEC SQL CONNECT TO testdb AS con2 USER testuser;
 EXEC SQL CONNECT TO testdb AS con3 USER testuser;
 EXEC SQL DISCONNECT CURRENT; /* Close connection 3. */
 EXEC SQL DISCONNECT DEFAULT; /* Close the default connection. */
 EXEC SQL DISCONNECT ALL; /* Close connections 2 and 1. */
 return 0;
}
```

#### Helpful Links

[CONNECT](#) and [SET CONNECTION](#)

### 5.8.5.5.7 EXECUTE IMMEDIATE

#### Function

Immediately prepares and executes a dynamically specified SQL statement without retrieving result rows.

## Syntax

```
EXECUTE IMMEDIATE string
```

## Parameter Description

### string

A C string or host variable that contains the SQL statement to be executed.

## Example

The following is an example of executing the INSERT statement using EXECUTE IMMEDIATE and a host variable named **command**:

```
sprintf(command, "INSERT INTO test (name, amount, letter) VALUES ('r1', 1, 'f');
EXEC SQL EXECUTE IMMEDIATE :command;
```

### 5.8.5.5.8 GET DESCRIPTOR

## Function

Retrieves information about a query result set and stores it into host variables. A descriptor area is typically populated using FETCH or SELECT before using this command to transfer the information into host language variables. This command can be in either of the following formats:

- Retrieves the descriptor "header" items, which applies to the result set in its entirety.
- Retrieves information about a particular column, requiring the column number as additional parameter.

## Syntax

```
GET DESCRIPTOR descriptor_name VALUE column_number :cvariable = descriptor_item [, ...]
GET DESCRIPTOR descriptor_name:cvariable = descriptor_header_item [, ...]
```

## Parameter Description

- **descriptor\_name**  
SQL descriptor name.
- **descriptor\_header\_item**  
Identifies which header item is to be retrieved. Currently, only COUNT that is used to obtain the number of columns in the result set is supported.
- **column\_number**  
Number of the column about which information is to be retrieved. The count starts at 1.
- **descriptor\_item**  
Identifies which information item about a column is to be retrieved.
- **cvariable**  
A host variable that will receive the data retrieved from the descriptor area.

## Example

Retrieve the number of columns in a result set.

```
EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
```

Retrieve the data length in the first column.

```
EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH;
```

Retrieve the data body of the second column as a string.

```
EXEC SQL GET DESCRIPTOR d VALUE 2 :d_data = DATA;
```

Execute **SELECT current\_database();**. The number of columns, column data length, and column data are displayed.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
 int d_count;
 char d_data[1024];
 int d_returned_octet_length;
EXEC SQL END DECLARE SECTION;
EXEC SQL CONNECT TO testdb AS con1 USER testuser;
EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
EXEC SQL ALLOCATE DESCRIPTOR d;

 /* Declare and open a cursor, and allocate a descriptor to the cursor. */
EXEC SQL DECLARE cur CURSOR FOR SELECT current_database();
EXEC SQL OPEN cur;
EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;

 /* Obtain the total number of columns. */
EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
printf("d_count = %d\n", d_count);

 /* Obtain the length of a returned column. */
EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH;
printf("d_returned_octet_length = %d\n", d_returned_octet_length);

 /* Fetch the returned column as a string. */
EXEC SQL GET DESCRIPTOR d VALUE 1 :d_data = DATA;
printf("d_data = %s\n", d_data);

 /* Closed */
EXEC SQL CLOSE cur;
EXEC SQL COMMIT;

EXEC SQL DEALLOCATE DESCRIPTOR d;
EXEC SQL DISCONNECT ALL;
return 0;
}
```

The following is the command output.

```
d_count = 1
d_returned_octet_length = 6
d_data = testdb
```

## Helpful Links

[ALLOCATE DESCRIPTOR](#), [DEALLOCATE DESCRIPTOR](#), and [SET DESCRIPTOR](#)

### 5.8.5.5.9 OPEN

#### Description

Opens a cursor and optionally binds actual values to placeholders in the cursor declaration. The cursor must have been declared using the **DECLARE** command. Executing the **OPEN** command triggers the query on the server.

#### Syntax

```
OPEN cursor_name
OPEN cursor_name USING value [, ...]
OPEN cursor_name USING SQL_DESCRIPTOR descriptor_name
```

#### Parameters

- **cursor\_name**  
Name of the cursor to be opened. It can be an SQL identifier or a host variable.
- **value**  
A value that is to be bound to a placeholder in the cursor declaration. It can be an SQL constant, a host variable, or a host variable with an indicator.
- **descriptor\_name**  
Name of the descriptor that contains the value to be bound to the placeholder in the cursor declaration. It can be an SQL identifier or a host variable.

#### Examples

```
EXEC SQL OPEN a;
EXEC SQL OPEN d USING 1, 'test';
EXEC SQL OPEN c1 USING SQL_DESCRIPTOR mydesc;
EXEC SQL OPEN :curname1;
```

#### Helpful Links

[DECLARE](#)

### 5.8.5.5.10 PREPARE

#### Description

Prepares the statement to be executed.

#### Syntax

```
PREPARE name FROM string
```

#### Parameters

- **name**  
An identifier for the prepared query.
- **string**

A C string or host variable that contains a prepared statement, which can be SELECT, INSERT, UPDATE, or DELETE.

## Examples

```
char *stmt = "SELECT * FROM test1 WHERE a = ? AND b = ?";
EXEC SQL ALLOCATE DESCRIPTOR outdesc;
EXEC SQL PREPARE foo FROM :stmt;
EXEC SQL EXECUTE foo USING SQL DESCRIPTOR indesc INTO SQL DESCRIPTOR outdesc;
```

### NOTICE

The PREPARE statement provided by ecpg is not equivalent to the PREPARE syntax provided by the kernel. Example:

GaussDB Kernel kernel syntax:

```
PREPARE name [(data_type [, ...])] AS statement
```

Embedded SQL statement:

```
EXEC SQL PREPARE I(int, int) AS INSERT INTO T VALUES ($1, $2);
EXEC SQL EXECUTE I(1, 2);
```

When the preceding statement is executed, an error message "too few arguments on" is reported. ecpg provides a dynamic SQL statement to solve the problem in the **PREPARE name [ ( data\_type [, ...] ) ] AS statement** syntax scenario.

Example of using dynamic SQL syntax rules to solve the preceding problem:

```
EXEC SQL PREPARE I AS INSERT INTO T VALUES ($1, $2);
EXEC SQL EXECUTE I using 1, 2;
```

## 5.8.5.5.11 SET AUTOCOMMIT

### Function

Sets the autocommit behavior of the current database session. By default, embedded SQL programs do not automatically commit, so you need to explicitly issue COMMIT. This command can change the session to the automatic commit mode so that each individual statement is implicitly committed.

### Syntax

```
SET AUTOCOMMIT { = | TO } { ON | OFF }
```

## 5.8.5.5.12 SET CONNECTION

### Function

Sets a database connection.

### Syntax

```
SET CONNECTION [TO | =] connection_name
```

### Parameter Description

- **connection\_name**  
Name of a database connection established by the CONNECT command.

## Example

```
EXEC SQL SET CONNECTION TO con2;
EXEC SQL SET CONNECTION = con1;
```

## Helpful Links

[CONNECT](#) and [DISCONNECT](#)

### 5.8.5.5.13 SET DESCRIPTOR

## Description

Populates an SQL descriptor area, which is usually used to bind parameters in a prepared query execution. This command can be in either of the following formats:

- Applies to the descriptor "header", which is independent of specific data.
- Assigns a value to specific data identified by a number.

## Syntax

```
SET DESCRIPTOR descriptor_name descriptor_header_item = value [, ...]
SET DESCRIPTOR descriptor_name VALUE number descriptor_item = value [, ...]
```

## Parameters

- **descriptor\_name**  
SQL descriptor name.
- **descriptor\_header\_item**  
Identifies the header information item to be set. Currently, only COUNT that can be used to set the number of descriptor items is supported.
- **number**  
Number of descriptor items to be set. The count starts at 1.
- **descriptor\_item**  
Identifies which descriptor item is to be set. Currently, only DATA, TYPE, and LENGTH are supported.
- **value**  
Value to be stored in the descriptor item. The value can be an SQL constant or a host variable.

## Examples

```
EXEC SQL SET DESCRIPTOR indesc COUNT = 1;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = 2;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = :val1;
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val1, DATA = 'some string';
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val2null, DATA = :val2;
```

## Helpful Links

[ALLOCATE DESCRIPTOR](#), [DEALLOCATE DESCRIPTOR](#), and [GET DESCRIPTOR](#)

### 5.8.5.5.14 TYPE

#### Function

Defines a new data type. This command is identified only when ecpg is run with the `-c` option.

#### Syntax

```
TYPE type_name IS ctype
```

#### Parameter Description

**type\_name**

Data type name

**ctype**

C type description.

#### Example

```
EXEC SQL TYPE customer IS
struct
{
 varchar name[50];
 int phone;
};

EXEC SQL TYPE cust_ind IS
struct ind
{
 short name_ind;
 short phone_ind;
};

EXEC SQL TYPE c IS char reference;
EXEC SQL TYPE ind IS union { int integer; short smallint; };
EXEC SQL TYPE intarray IS int[AMOUNT];
EXEC SQL TYPE str IS varchar[BUFFERSIZ];
EXEC SQL TYPE string IS char[11];
```

Example of using EXEC SQL TYPE (note that the `-c` parameter needs to be added in the ecpg preprocessing phase when the following example is used):

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

EXEC SQL WHENEVER SQLERROR SQLPRINT;
EXEC SQL TYPE tt IS
struct
{
 varchar v[256];
 int i;
};
EXEC SQL TYPE tt_ind IS
struct ind {
 short v_ind;
 short i_ind;
};

int main(void)
{
```

```
EXEC SQL BEGIN DECLARE SECTION;
 tt t;
 tt_ind t_ind;
EXEC SQL END DECLARE SECTION;
EXEC SQL CONNECT TO testdb AS con1;
EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
EXEC SQL SELECT current_database(), 256 INTO :t_ind LIMIT 1;

printf("t.v = %s\n", t.v.arr);
printf("t.i = %d\n", t.i);

printf("t_ind.v_ind = %d\n", t_ind.v_ind);
printf("t_ind.i_ind = %d\n", t_ind.i_ind);

EXEC SQL DISCONNECT con1;

return 0;
}
```

The output of this example is as follows.

```
t.v = testdb
t.i = 256
t_ind.v_ind = 0
t_ind.i_ind = 0
```

### 5.8.5.5.15 VAR

## Function

Assigns a new C data type to a host variable. The host variable must have been declared in a DECLARE segment.

#### NOTE

- Exercise caution when using VAR. Using VAR to change the data type may cause the memory address to be invalid. As a result, the data variable is invalid and the value cannot be assigned.
- If the data type has been defined in the host variable DECLARE segment, you do not need to use the VAR statement.

## Syntax

```
VAR varname IS ctype
```

## Parameter Description

- **varname**  
Name of a C variable.
- **ctype**  
C type description.

## Example

```
EXEC SQL BEGIN DECLARE SECTION;
 short a;
EXEC SQL END DECLARE SECTION;
EXEC SQL VAR a IS int;
```

### 5.8.5.5.16 WHENEVER

#### Description

Defines a behavior that is called when an SQL execution exception occurs (row not found, SQL alarm, or error).

#### Syntax

```
WHENEVER { NOT FOUND | SQLERROR | SQLWARNING } action
```

#### Parameters

For details about the parameter description, see [Setting Callbacks](#).

#### Examples

```
EXEC SQL WHENEVER NOT FOUND CONTINUE;
EXEC SQL WHENEVER NOT FOUND DO BREAK;
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLWARNING DO warn();
EXEC SQL WHENEVER SQLERROR sqlprint;
EXEC SQL WHENEVER SQLERROR SQLCALL print2();
EXEC SQL WHENEVER SQLERROR DO handle_error("select");
EXEC SQL WHENEVER SQLERROR DO sqlnotice(NULL, NONO);
EXEC SQL WHENEVER SQLERROR DO sqlprint();
EXEC SQL WHENEVER SQLERROR GOTO error_label;
EXEC SQL WHENEVER SQLERROR STOP;
```

Use `WHENEVER NOT FOUND BREAK` to handle the looping of the result set.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
 EXEC SQL CONNECT TO testdb AS con1;
 EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
 EXEC SQL ALLOCATE DESCRIPTOR d;
 EXEC SQL DECLARE cur CURSOR FOR SELECT current_database(), 'hoge', 256;
 EXEC SQL OPEN cur;

 /* When the end of the result set is reached, exit the loop. */
 EXEC SQL WHENEVER NOT FOUND DO BREAK;

 while (1)
 {
 EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;
 exec sql get descriptor d value 1 :d1=DATA;
 exec sql get descriptor d value 2 :d2=DATA;
 printf("d1 is %s,%s\n", d1, d2) ;
 }
 EXEC SQL CLOSE cur;
 EXEC SQL COMMIT;
 EXEC SQL DEALLOCATE DESCRIPTOR d;
 EXEC SQL DISCONNECT ALL;
 return 0;
}
```

## 5.8.6 Querying the Result Set

- The `SELECT` statement that returns the result of a single row can be directly executed using `EXEC SQL`. For details, see [Running SQL Commands](#).

Example:

```
/* Create a table and insert data. */
EXEC SQL CREATE TABLE test_table (number1 integer, number2 integer);
EXEC SQL INSERT INTO test_table (number1, number2) VALUES (2, 1);

/* The query result is in a single row. :num is the host variable. */
EXEC SQL SELECT number1 INTO :num FROM test_table WHERE number2 = 1;
```

- To process a multi-row result set, you must use a cursor. For details, see [Using Cursors](#). (In special cases, an application can fetch multiple rows of results at a time and write them to the host variable of the array type. For details, see [Host Variables with Non-primitive Types](#).)

Example:

```
/* Create a table and insert data. */
EXEC SQL CREATE TABLE test_table (number1 integer, number2 integer);
EXEC SQL INSERT INTO test_table (number1, number2) VALUES (2, 1);
EXEC SQL INSERT INTO test_table (number1, number2) VALUES (3, 1);
EXEC SQL INSERT INTO test_table (number1, number2) VALUES (4, 1);
EXEC SQL INSERT INTO test_table (number1, number2) VALUES (5, 1);

/* Define the host variable. */
EXEC SQL BEGIN DECLARE SECTION;
int v1;
int v2;
EXEC SQL END DECLARE SECTION;

/* Declare a cursor. */
EXEC SQL DECLARE test_bar CURSOR FOR SELECT number1, number2 FROM test_table ORDER BY
number1;
/* Open the cursor. */
EXEC SQL OPEN test_bar;
/* When the cursor reaches the end of the result set, exit the loop. */
EXEC SQL WHENEVER NOT FOUND DO BREAK;
/* Obtain the query result set. */
while(1)
{
 EXEC SQL FETCH NEXT FROM test_bar INTO :v1, :v2;
 printf("number1 = %d, number2 = %d\n",v1,v2);
}
/* Close the cursor. */
EXEC SQL CLOSE test_bar;
```

## 5.8.7 Closing a Database Connection

Close the database connection after the database is used.

Close a connection.

```
EXEC SQL DISCONNECT [connection];
```

Declare the connection by using the following methods:

- connection-name
- default
- current
- all

## 5.8.8 Host Variables

This section describes how to use host variables to pass data between C programs and embedded SQL programs. In C programs with embedded SQL statements, we use the C language as the host language and regard EXEC SQL [Command] statements as embedded SQL statements of the host language. Therefore,

variables used for embedded SQL statements in C programs are called host variables.

### 5.8.8.1 Overview

Passing data between a C program and SQL statements is particularly simple in embedded SQL. Instead of having the program paste data into the statements, you can simply write the name of a C variable into the SQL statement, prefixed by a colon. The following is an example.

```
EXEC SQL INSERT INTO sometable VALUES (:v1, 'foo', :v2);
```

This statement references two C variables named v1 and v2 and uses a regular SQL string, indicating that you are not restricted to use one kind of data or the other.

### 5.8.8.2 DECLARE Section

To implement data interaction between a C program with embedded SQL statements and a database, for example, to pass parameters in a query of the C program to the database, or to pass data from the database back to the program, the C variables that are intended to contain this data need to be declared in specially marked sections, so that the embedded SQL preprocessor is made aware of them.

This section starts with:

```
EXEC SQL BEGIN DECLARE SECTION;
```

And ends with:

```
EXEC SQL END DECLARE SECTION;
```

Between them, there must be regular C variable declarations. An example is as follows:

```
int x = 4;
char foo[16], bar[16];
```

#### NOTICE

- The type of the host variables declared between the start and end of the marked section must be one of the supported data types. For details, see [Table 5-88](#).
- You can also declare variables with the following syntax which implicitly creates a DECLARE section: EXEC SQL int i = 4.
- Variables that are not intended to be used in SQL commands can be declared normally outside these special sections.
- The definition of a structure or union also must be listed inside a DECLARE section. Otherwise, the preprocessor cannot process these types.

### 5.8.8.3 Retrieving Query Results

To retrieve the results of a query, embedded SQL provides special variants of the usual commands **SELECT** and **FETCH**. These commands have a special INTO clause that specifies which host variables the retrieved values are to be stored in. **SELECT** is used for a query that returns only a single row, and **FETCH** is used for a query that returns multiple rows, using a cursor.

- Use SELECT.

```

/*
 * Assume a table:
 * CREATE TABLE test1 (a int, b varchar(50));
 */
EXEC SQL BEGIN DECLARE SECTION;
 int v1;
 VARCHAR v2;
EXEC SQL END DECLARE SECTION;

...

EXEC SQL SELECT a, b INTO :v1, :v2 FROM test;

```

The INTO clause appears between the SELECT list and the FROM clause. The number of elements in the SELECT list and the list after INTO (also called the target list) must be equal.

- Use FETCH.

```

EXEC SQL BEGIN DECLARE SECTION;
 int v1;
 VARCHAR v2;
EXEC SQL END DECLARE SECTION;

...

EXEC SQL DECLARE foo CURSOR FOR SELECT a, b FROM test;
...
do
{
 ...
 EXEC SQL FETCH NEXT FROM foo INTO :v1, :v2;
 ...
} while (...);

```

The INTO clause appears after all SQL clauses.

### 5.8.8.4 Type Mapping

When ecpg applications exchange values between the GaussDB Kernel server and the C program, such as when retrieving query results from the server or executing SQL statements with input parameters, the values need to be converted between GaussDB Kernel data types and host language variable types (C language data types, concretely). There are two data types available: Simple GaussDB Kernel data types, such as integer and text, can be directly read and written by applications. Other GaussDB Kernel data types, such as timestamp and numeric, can be accessed only by using special library functions. For details, see [ECPG API Reference](#).

**Table 5-88** Mapping between GaussDB Kernel data types and C variable types

| GaussDB Kernel Data Type       | Host Variable Type      |
|--------------------------------|-------------------------|
| smallint                       | short                   |
| integer                        | int                     |
| bigint                         | long long int           |
| boolean                        | boolean                 |
| character(n), varchar(n), text | char[n+1], VARCHAR[n+1] |
| double precision               | double                  |

| GaussDB Kernel Data Type | Host Variable Type |
|--------------------------|--------------------|
| real                     | float              |
| smallserial              | short              |
| serial                   | int                |
| bigserial                | long long int      |
| oid                      | unsigned int       |
| name                     | char[NAMEDATALEN]  |
| date                     | date [a]           |
| timestamp                | timestamp [a]      |
| interval                 | interval [a]       |
| decimal                  | decimal [a]        |
| numeric                  | numeric [a]        |

 **NOTE**

[a] This type can be accessed through [Accessing Special Data Types](#).

**NOTICE**

- Currently, only basic data types of the C language can be used or combined. The string data type in the C++ language cannot be used as the host variable type.
- Currently, ecpg maps only common data types of GaussDB Kernel SQL. For details about the supported data types, see [Table 5-88](#).

### 5.8.8.5 Handling Character Strings

To handle SQL character string data types, such as varchar and text, there are two possible methods to declare the host variables.

1. Method 1: Use char[] (a char string), which is the most common method for processing character data in C programs.

```
EXEC SQL BEGIN DECLARE SECTION;
char str[50];
EXEC SQL END DECLARE SECTION;
```

Note that you have to take care of the length yourself. If you use this host variable as the target variable of a query which returns a string with more than 49 characters, a buffer overflow occurs.

2. Method 2: Use the VARCHAR type, which is a special type provided by ecpg. The definition on an array of type VARCHAR is converted into a named struct for every variable. The statement is as follows:

```
VARCHAR var[180];
```

It will be converted into:

```
struct varchar_var
{
 int len;
 char arr[180];
} var;
```

To store a string in a VARCHAR host variable, the host variable has to be declared as a string including the zero-byte terminator. **arr** stores the string including a terminating zero byte. **len** stores the length of the string stored in **arr** without the terminating zero byte. The terminator is not included when the length is calculated. When a host variable is used as input for a query, if the values of **strlen(arr)** and **len** are different, the shorter one is used.

---

**CAUTION**

- VARCHAR can be written in upper or lower case, but not in mixed case.
  - char and VARCHAR host variables can also hold values of other SQL types, which will be stored in their string forms.
- 

### 5.8.8.6 Host Variables with Non-primitive Types

Non-primitive host variables can be arrays, typedefs, structures, and pointers.

- Arrays

There are two use cases for arrays as host variables. The first case is to store some text strings in `char[]` or `VARCHAR[]`. The second case is to retrieve multiple rows from a query result without using a cursor. Without an array, to process a query result consisting of multiple rows, it is required to use a cursor and the **FETCH** command. But with array host variables, multiple rows can be retrieved at once. The length of the array has to be defined to be able to accommodate all rows; otherwise, a buffer overflow will occur.

For example, scan the `pg_database` system catalog and display the OIDs and names of all available databases.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
 EXEC SQL BEGIN DECLARE SECTION;
 int dbid[8];
 char dbname[8][16];
 int i;
 EXEC SQL END DECLARE SECTION;

 memset(dbname, 0, sizeof(char)* 16 * 8);
 memset(dbid, 0, sizeof(int) * 8);
 * Connect to the testdb database. The testdb database must be created in advance. */
 EXEC SQL CONNECT TO testdb;
 /* Retrieve multiple rows to arrays at a time. */
 EXEC SQL SELECT oid,datname INTO :dbid, :dbname FROM pg_database;
 for (i = 0; i < 8; i++)
 printf("oid=%d, dbname=%s\n", dbid[i], dbname[i]);
 EXEC SQL COMMIT;
 EXEC SQL DISCONNECT ALL;
 return 0;
}
```

Example output (the exact value depends on the local environment):

```
oid=1, dbname=template1
oid=11510, dbname=template0
oid=11511, dbname=postgres
oid=313780, dbname=testdb
oid=0, dbname=
oid=0, dbname=
oid=0, dbname=
```

- Structures

A structure whose member names match the column names of a query result, can be used to retrieve multiple columns at once. The structure enables handling multiple column values in a single host variable.

The following example retrieves OIDs, names, and sizes of the available databases from the `pg_database` system catalog and using the `pg_database_size()` function. In this example, a structure variable with members whose names match each column in the `SELECT` result is used to retrieve one result row without putting multiple host variables in the `FETCH` statement.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct
{
 int oid;
 char datname[65];
 long long int size;
} dbinfo_t;

dbinfo_t dbval;
EXEC SQL END DECLARE SECTION;
memset(&dbval, 0, sizeof(dbinfo_t));

EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size FROM
pg_database;
EXEC SQL OPEN cur1;

/* Exit the while loop when the end of the result set is reached. */
EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
 /* Fetch multiple columns into one structure. */
 EXEC SQL FETCH FROM cur1 INTO :dbval;
 /* Print the members of the structure. */
 printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.datname, dbval.size);
}
EXEC SQL CLOSE cur1;
```

Example output (the exact value depends on the local environment):

```
oid=1, datname=template1, size=4324580
oid=11510, datname=template0, size=4243460
oid=11511, datname=postgres, size=4324580
oid=313780, datname=testdb, size=8183012
```

Structure host variables "absorb" as many columns in the query result as the structure columns. Additional columns can be allocated to other host variables. The above program could also be restructured like this, with the **size** variable outside the structure:

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct
{
 int oid;
 char datname[65];
} dbinfo_t;

dbinfo_t dbval;
```

```
long long int size;
EXEC SQL END DECLARE SECTION;

memset(&dbval, 0, sizeof(dbinfo_t));

EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size FROM
pg_database;
EXEC SQL OPEN cur1;

/* Exit the while loop when the end of the result set is reached. */
EXEC SQL WHENEVER NOT FOUND DO BREAK;
while (1)
{
/* Fetch multiple columns into one structure. */
EXEC SQL FETCH FROM cur1 INTO :dbval, :size;
/* Print the members of the structure. */
printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.datname, size);
}

EXEC SQL CLOSE cur1;
```

- **typedef**

Use the `typedef` keyword to map new types to existing types.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef char mychartype[40];
typedef long serial_t;
EXEC SQL END DECLARE SECTION;
```

You can also run the following command:

```
EXEC SQL TYPE serial_t IS long;
```

This declaration does not need to be part of a `DECLARE` section.

- **Pointers**

You can declare pointers to the most common types.

```
EXEC SQL BEGIN DECLARE SECTION;
int *intp;
char **charp;
EXEC SQL END DECLARE SECTION;
```

### 5.8.8.7 Accessing Special Data Types

ecpg supports the numeric, decimal, date, timestamp, and interval data types. These data types cannot be mapped to primitive host variable types because they have a complex internal structure. Applications deal with these types by declaring host variables in special types and accessing them using functions in the `pgtypes` library. For details about the functions in the `pgtypes` library, see [ECPG API Reference](#).

- **timestamp,date**

First, the program must include the header file for the timestamp type.

```
#include <pgtypes_timestamp.h>
```

Then, declare a host variable of the timestamp type in the `DECLARE` section.

```
EXEC SQL BEGIN DECLARE SECTION;
timestamp ts;
EXEC SQL END DECLARE SECTION;
```

After the value is read to the host variable, the `pgtypes` library function is used for processing. In the following example, the `PGTYPEStimestamp_to_asc()` function is used to convert the timestamp value to the text format:

```
EXEC SQL SELECT now()::timestamp INTO :ts;
printf("ts = %s\n", PGTYPEStimestamp_to_asc(ts));
```

Example output:

```
ts = 2022-06-27 18:03:56.949343
```

In addition, the date type can be processed in the same way. The program must contain the **pgtypes\_data.h** header file, declare a host variable as the date type, and use the `PGTYPEdata_to_asc()` function to convert the variable to the text format.

- interval

The handling of the interval type is also similar to the timestamp and date types. However, to allocate memory for an interval type value explicitly, the memory space for the variable must be allocated from the heap memory.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_interval.h>
int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
 interval *in;
EXEC SQL END DECLARE SECTION;
 /* Connect to the testdb database. The testdb database must be created in advance.*/
EXEC SQL CONNECT TO testdb;
 in = PGTYPEinterval_new();
EXEC SQL SELECT '1 min'::interval INTO :in;
 printf("interval = %s\n", PGTYPEinterval_to_asc(in));
 PGTYPEinterval_free(in);
EXEC SQL COMMIT;
EXEC SQL DISCONNECT ALL;
 return 0;
}
```

- numeric,decimal

The handling of the numeric and decimal types is similar to the interval type: It requires defining a pointer, allocating some memory space from the heap, and accessing the variable using the `pgtypes` library functions.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_numeric.h>
EXEC SQL WHENEVER SQLERROR STOP;
int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
 numeric *num;
 numeric *num2;
 decimal *dec;
EXEC SQL END DECLARE SECTION;

 /* Connect to the testdb database. The testdb database must be created in advance.*/
EXEC SQL CONNECT TO testdb;

 num = PGTYPEnumeric_new();
 dec = PGTYPEdecimal_new();

EXEC SQL SELECT 12.345::numeric(4,2), 23.456::decimal(4,2) INTO :num, :dec;
 printf("numeric = %s\n", PGTYPEnumeric_to_asc(num, 0));
 printf("numeric = %s\n", PGTYPEnumeric_to_asc(num, 1));
 printf("numeric = %s\n", PGTYPEnumeric_to_asc(num, 2));
 /* Convert decimal to numeric to show a decimal value. */
 num2 = PGTYPEnumeric_new();
 PGTYPEnumeric_from_decimal(dec, num2);
 printf("decimal = %s\n", PGTYPEnumeric_to_asc(num2, 0));
 printf("decimal = %s\n", PGTYPEnumeric_to_asc(num2, 1));
 printf("decimal = %s\n", PGTYPEnumeric_to_asc(num2, 2));
 PGTYPEnumeric_free(num2);
 PGTYPEdecimal_free(dec);
}
```

```
PGTYPESnumeric_free(num);

EXEC SQL COMMIT;
EXEC SQL DISCONNECT ALL;
return 0;
}
```

### 5.8.8.8 Handling Non-primitive SQL Data Types

This section describes how to handle non-scalar and user-defined SQL-level data types in ecpg applications. Note that this is distinct from the handling of host variables of non-primitive types described in [Host Variables with Non-primitive Types](#).

- Arrays

Multi-dimensional SQL-level arrays are not directly supported in ecpg. One-dimensional SQL-level arrays can be mapped into C array host variables and vice-versa. However, when creating a statement, ecpg does not know the types of the columns, so that it cannot check if a C array is input into a corresponding SQL-level array. When processing the output of an SQL statement, ecpg has to check if both are arrays.

Example:

```
CREATE TABLE t3 (
 ii integer[]
);
testdb=> SELECT * FROM t3;
 ii

{1,2,3,4,5}
(1 row)
```

The following example retrieves the fourth element of an array and stores it in a host variable of the int type:

```
EXEC SQL BEGIN DECLARE SECTION;
 int ii;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[4] FROM t3;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
 EXEC SQL FETCH FROM cur1 INTO :ii ;
 printf("ii=%d\n", ii);
}
EXEC SQL CLOSE cur1;
```

Example output:

```
ii=4
```

To map multiple array elements to the multiple elements in an array type host variable, each element of the array column and each element of the host variable array must be managed separately. Example:

```
EXEC SQL BEGIN DECLARE SECTION;
 int ii_a[8];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[1], ii[2], ii[3], ii[4] FROM t3;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;
while (1)
```

```
{
 EXEC SQL FETCH FROM cur1 INTO :ii_a[0], :ii_a[1], :ii_a[2], :ii_a[3];
 ...
}
```

**Note:**

```
EXEC SQL BEGIN DECLARE SECTION;
 int ii_a[8];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii FROM t3;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
 /* Error */
 EXEC SQL FETCH FROM cur1 INTO :ii_a;
 ...
}
```

It does not work out because you cannot map an array type column to an array host variable directly.

- Composite types

Composite types are not directly supported in ecpg. For example, you cannot declare member variables as date type in a structure. However, you can access each attribute separately or use the external string representation.

In the following example, each attribute can be accessed separately:

```
CREATE TYPE comp_t AS (intval integer, textval varchar(32));
CREATE TABLE t4 (compval comp_t);
INSERT INTO t4 VALUES ((256, 'PostgreSQL'));
```

The following program retrieves data from the example table by selecting each attribute of the `comp_t` type separately:

```
EXEC SQL BEGIN DECLARE SECTION;
 int intval;
 varchar textval[33];
EXEC SQL END DECLARE SECTION;

/* Put each element of the composite type column in the SELECT list. */
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;
while (1)
{
 /* Fetch each element of the composite type column into host variables. */
 EXEC SQL FETCH FROM cur1 INTO :intval, :textval;
 printf("intval=%d, textval=%s\n", intval, textval.arr);
}
EXEC SQL CLOSE cur1;
```

The host variables storing values in the **FETCH** command can be gathered into one structure. For more details about the host variables in the structure form, see [Handling Character Strings](#). In the following example, the two host variables, `intval` and `textval`, become members of the `comp_t` structure, and the structure is specified in the **FETCH** command:

```
EXEC SQL BEGIN DECLARE SECTION;
 typedef struct
 {
 int intval;
 varchar textval[33];
 } comp_t;
 comp_t compval;
EXEC SQL END DECLARE SECTION;
```

```
/* Put each element of the composite type column in the SELECT list. */
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;
while (1)
{
 /* Put all values in the SELECT list into one structure. */
 EXEC SQL FETCH FROM cur1 INTO :compval;
 printf("intval=%d, textval=%s\n", compval.intval, compval.textval.arr);
}
EXEC SQL CLOSE cur1;
```

Although a structure is used in the **FETCH** command, the attribute names in the **SELECT** clause are specified one by one. This can be enhanced by using a **\*** to ask for all attributes of the composite type value. For example:

```
...
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).* FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
 /* Put all values in the SELECT list into one structure. */
 EXEC SQL FETCH FROM cur1 INTO :compval;
 printf("intval=%d, textval=%s\n", compval.intval, compval.textval.arr);
}
...
```

In this way, composite types can be mapped into structures, even though `ecpg` does not understand the composite type itself.

- User-defined base types

When `ecpg` uses host variables to store query results, only the data types provided by `ecpg` are supported. Data types created using `CREATE TYPE` cannot be mapped using host variables.

The external string representation of that type is `(%lf,%lf)`, which is defined in function `complex_in()`. The following example inserts complex type values **(1,1)** and **(3,3)** into columns **a** and **b** and then queries them from the table:

```
EXEC SQL BEGIN DECLARE SECTION;
 varchar a[64];
 varchar b[64];
EXEC SQL END DECLARE SECTION;
EXEC SQL INSERT INTO test_complex VALUES ('(1,1)', '(3,3)');
EXEC SQL DECLARE cur1 CURSOR FOR SELECT a, b FROM test_complex;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
 EXEC SQL FETCH FROM cur1 INTO :a, :b;
 printf("a=%s, b=%s\n", a.arr, b.arr);
}
EXEC SQL CLOSE cur1;
```

Example output:  
a=(1,1), b=(3,3)

## 5.8.9 Executing Dynamic SQL Statements

In most cases, the SQL statements executed by an application are known when the application is written. However, in some cases, SQL statements are constructed at run time or provided by external sources. In these cases, the SQL statements

cannot be directly embedded into the C source code, but dynamic SQL statements allow you to call the provided SQL statements in a string variable.

### 5.8.9.1 Executing a Statement Without a Result Set

An example of running the **EXECUTE IMMEDIATE** command is as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
 const char *stmt = "CREATE TABLE test1 (...);";
EXEC SQL END DECLARE SECTION;
EXEC SQL EXECUTE IMMEDIATE :stmt;
```

**EXECUTE IMMEDIATE** can be used for SQL statements that do not return a result set, such as DDL, INSERT, UPDATE, and DELETE statements. However, statements for retrieving data, such as SELECT statements, cannot be executed in this way.

### 5.8.9.2 Executing a Statement with Input Parameters

Prepare a normal statement and execute a specific version of it by replacing its parameters (with question marks). Use the EXECUTE statement to execute the prepared statement by specifying parameters in the USING clause. Example:

```
EXEC SQL BEGIN DECLARE SECTION;
 const char *stmt = "INSERT INTO test1 VALUES(?, ?);";
EXEC SQL END DECLARE SECTION;
/* PREPARE Prepare a statement for execution. */
EXEC SQL PREPARE mystmt FROM :stmt;
...
/* Single quotation marks are valid characters. If a character string is used, use double quotation marks. */
EXEC SQL EXECUTE mystmt USING 42, 'foobar';

/* When you no longer need a prepared statement, you should deallocate it. */
EXEC SQL DEALLOCATE PREPARE name;
```

### 5.8.9.3 Executing a Statement with a Result Set

EXECUTE can be used to execute SQL statements with a result set. To save the result, add an INTO clause. Example:

```
EXEC SQL BEGIN DECLARE SECTION;
 const char *stmt = "SELECT a, b, c FROM test1 WHERE a > ?";
 int v1, v2;
 VARCHAR v3[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
...
EXEC SQL EXECUTE mystmt INTO :v1, :v2, :v3 USING 37;
```

#### NOTE

The EXECUTE statement supports the INTO and USING clauses.

If a query may return multiple result rows, use cursors. For details about cursors, see [Using Cursors](#). Example:

```
EXEC SQL BEGIN DECLARE SECTION;
 char dbaname[128];
 char datname[128];
 char *stmt = "SELECT u.username as dbaname, d.datname "
 " FROM pg_database d, pg_user u "
 " WHERE d.datdba = u.usesysid";
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO testdb AS con1 USER testuser;

EXEC SQL PREPARE stmt1 FROM :stmt;
```

```
EXEC SQL DECLARE cursor1 CURSOR FOR stmt1;
EXEC SQL OPEN cursor1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
 EXEC SQL FETCH cursor1 INTO :dbname,:datname;
 printf("dbname=%s, datname=%s\n", dbname, datname);
}
EXEC SQL CLOSE cursor1;

EXEC SQL COMMIT;
EXEC SQL DISCONNECT ALL;
```

## 5.8.10 Error Handling

There are two non-exclusive methods to handle errors and warnings in an embedded SQL program:

- Callbacks can be configured to handle errors or warnings using the `WHENEVER` command.
- Detailed information about the errors or warnings can be obtained from the `sqlca` variable.

### 5.8.10.1 Setting Callbacks

One simple method to catch errors and warnings is to set a specific action to be executed whenever a particular condition occurs. To set the callback, run the following command:

```
EXEC SQL WHENEVER condition action;
```

**condition** can be one of the following:

- **SQLERROR**: The specified action is called whenever an error occurs during the execution of an SQL statement.
- **SQLWARNING**: The specified action is called whenever a warning occurs during the execution of an SQL statement.
- **NOT FOUND**: The specified action is called whenever an SQL statement retrieves or affects zero rows.

**action** can be one of the following:

- **CONTINUE**: ignores the callback error condition and continues the execution. It is usually used to stop a break condition. This is the default value.
- **GOTO label/GO TO label**: jumps to a specified label (using the C goto statement).
- **SQLPRINT**: prints a message to the standard error.
- **STOP**: calls `exit(1)` to terminate the program.
- **DO BREAK**: executes the C statement `BREAK`. This statement is used only in loops or switch statements.

Example:

```
/* It prints a simple message when a warning occurs and aborts the program when an error happens. */
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLERROR STOP;
```

**NOTICE**

- The statement EXEC SQL WHENEVER is a directive of the SQL preprocessor, not a C statement. The error or warning actions that it sets apply to all embedded SQL statements that appear below the point where the handler is set, unless a different action was set for the same condition between the first EXEC SQL WHENEVER and the SQL statement causing the condition, regardless of the flow of control in the C program. Therefore, neither of the following C programs can achieve the expected effect:

```
/*
 * ERROR
 */
int main(int argc, char *argv[])
{
 ...
 if (verbose) {
 EXEC SQL WHENEVER SQLWARNING SQLPRINT;
 }
 ...
 EXEC SQL SELECT ...;
 ...
}
/*
 * ERROR
 */
int main(int argc, char *argv[])
{
 ...
 set_error_handler();
 ...
 EXEC SQL SELECT ...;
 ...
}
static void set_error_handler(void)
{
 EXEC SQL WHENEVER SQLERROR STOP;
}
```

- DO BREAK can be used only in the while, for, and switch scenarios. After DO BREAK is used, use the CONTINUE statement to ignore it.

### 5.8.10.2 sqlca

The embedded SQL API provides a global variable *sqlca* (short for SQL communication area). *sqlca* covers warnings and errors. If multiple warnings or errors occur during the execution of a statement, then *sqlca* will only contain information about the last one. In a multithreaded program, every thread automatically gets its own copy of *sqlca*.

The data structure is as follows:

```
struct
{
 char sqlcaid[8];
 long sqlabc;
 long sqlcode;
 struct
 {
 int sqlerrm;
 char sqlerrmc[SQLERRMC_LEN];
 } sqlerrm;
 char sqlerrp[8];
 long sqlerrd[6];
 char sqlwarn[8];
}
```

```
char sqlstate[5];
} sqlca;
```

If no error occurred in the last SQL statement, **sqlca.sqlcode** will be **0** and **sqlca.sqlstate** will be **00000**. If a warning or error occurred, then **sqlca.sqlcode** will be negative and **sqlca.sqlstate** will be different from **00000**. For details about the values of **SQLSTATE** and **SQLCODE**, see [SQLSTATE and SQLCODE](#).

If the last SQL statement was successful, then **sqlca.sqlerrd[1]** contains the OID of the processed row, if applicable, and **sqlca.sqlerrd[2]** contains the number of processed or returned rows, if applicable to the command.

In case of an error or warning, **sqlca.sqlerrm.sqlerrmc** will contain a string that describes the error. **sqlca.sqlerrm.sqlerrml** contains the length of the error message that is stored in **sqlca.sqlerrm.sqlerrmc**, that is, the result of **strlen()**. Note that some messages are too long to fit in the fixed-size **sqlerrmc** array; they will be truncated.

When a warning is generated, **sqlca.sqlwarn[2]** is set to **W**.

The columns **sqlcaid**, **sqlcabc**, **sqlerrp**, and the remaining elements of **sqlerrd** and **sqlwarn** currently contain no useful information.

Example:

```
/* Integrate WHENEVER and sqlca to implement error handling. */
EXEC SQL WHENEVER SQLERROR SQLCALL print_sqlca();

void print_sqlca()
{
 fprintf(stderr, "==== sqlca ====\n");
 fprintf(stderr, "sqlcode: %ld\n", sqlca.sqlcode);
 fprintf(stderr, "sqlerrm.sqlerrml: %d\n", sqlca.sqlerrm.sqlerrml);
 fprintf(stderr, "sqlerrm.sqlerrmc: %s\n", sqlca.sqlerrm.sqlerrmc);
 fprintf(stderr, "sqlerrd: %ld %ld %ld %ld %ld %ld\n", sqlca.sqlerrd[0], sqlca.sqlerrd[1], sqlca.sqlerrd[2],
 sqlca.sqlerrd[3], sqlca.sqlerrd[4], sqlca.sqlerrd[5]);
 fprintf(stderr, "sqlwarn: %d %d %d %d %d %d %d %d\n", sqlca.sqlwarn[0], sqlca.sqlwarn[1],
 sqlca.sqlwarn[2],
 sqlca.sqlwarn[3], sqlca.sqlwarn[4], sqlca.sqlwarn[5],
 sqlca.sqlwarn[6], sqlca.sqlwarn[7]);
 fprintf(stderr, "sqlstate: %5s\n", sqlca.sqlstate);
 fprintf(stderr, "=====\n");
}
```

The output is similar to the following (here is a misspelled table name):

```
==== sqlca ====
sqlcode: -400
sqlerrm.sqlerrml: 49
sqlerrm.sqlerrmc: relation "pg_databasep" does not exist on line 38
sqlerrd: 0 0 0 0 0
sqlwarn: 0 0 0 0 0 0 0
sqlstate: 42P01
=====
```

### 5.8.10.3 SQLSTATE and SQLCODE

SQLSTATE is a five-character array. The five characters contain digits or upper-case letters that represent codes of various error and warning conditions. SQLSTATE has a hierarchical scheme: the first two characters indicate the general class of the condition, the last three characters indicate a subclass of the general condition. For example, the code 00000 indicates the success state.

SQLCODE is a simple integer. The value **0** indicates success, a positive value indicates success with additional information, and a negative value indicates an

error. The SQL standard only defines the positive value + 100, which indicates that the last command returned or affected zero rows, and no specific negative values.

**Table 5-89** Mapping between SQLSTATE and SQLCODE

| SQLCODE Value                     | SQLSTATE Value                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------------------|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0<br>(ECPG_NO_ERROR)              | SQLSTATE<br>00000             | Indicates no error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 100<br>(ECPG_NOT_FOUND)           | SQLSTATE<br>02000             | <p>This is a harmless condition indicating that the last command retrieved or processed zero rows, or that you are at the end of the cursor.</p> <p>When processing a cursor in a loop, you could use this code as a way to detect when to abort the loop. An example is as follows:</p> <pre>while (1) { EXEC SQL FETCH ... ; if (sqlca.sqlcode == ECPG_NOT_FOUND) break; }</pre> <p>Actually, WHENEVER NOT FOUND DO BREAK effectively does this internally, so there is usually no advantage in writing this out explicitly.</p> |
| -12<br>(ECPG_OUT_OF_MEMORY)       | SQLSTATE<br>YE001             | Indicates that your virtual memory is exhausted. The numeric value is defined as <b>-ENOMEM</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| -200<br>(ECPG_UNSUPPORTED)        | SQLSTATE<br>YE000             | Indicates that the preprocessor has generated something that the library does not know about.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| -201<br>(ECPG_TOO_MANY_ARGUMENTS) | SQLSTATE<br>07001 or<br>07002 | Indicates that the command specified more host variables than the command expected.                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| -202<br>(ECPG_TOO_FEW_ARGUMENTS)  | SQLSTATE<br>07001 or<br>07002 | Indicates that the command specified fewer host variables than the command expected.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| -203<br>(ECPG_TOO_MANY_MATCHES)   | SQLSTATE<br>21000             | Indicates that a query has returned multiple rows, but the statement is ready to store only one result row.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| -204<br>(ECPG_INT_FORMAT)         | SQLSTATE<br>42804             | The host variable is of the int type and the data in the database is of a different type and contains a value that cannot be interpreted as an int. The library uses strtol() for this conversion.                                                                                                                                                                                                                                                                                                                                 |

| SQLCODE Value                    | SQLSTATE Value    | Description                                                                                                                                                                                                           |
|----------------------------------|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -205<br>(ECPG_UINT_FORMAT)       | SQLSTATE<br>42804 | The host variable is of the unsigned int type and the data in the database is of a different type and contains a value that cannot be interpreted as an unsigned int. The library uses strtoul() for this conversion. |
| -206<br>(ECPG_FLOAT_FORMAT)      | SQLSTATE<br>42804 | The host variable is of the float type and the data in the database is of another type and contains a value that cannot be interpreted as a float value. The library uses strtod() for this conversion.               |
| -207<br>(ECPG_NUMERIC_FORMAT)    | SQLSTATE<br>42804 | The host variable is of the numeric type and the data in the database is of another type and contains a value that cannot be interpreted as a numeric value.                                                          |
| -208<br>(ECPG_INTERVAL_FORMAT)   | SQLSTATE<br>42804 | The host variable is of the interval type and the data in the database is of another type and contains a value that cannot be interpreted as an interval value.                                                       |
| -209<br>(ECPG_DATE_FORMAT)       | SQLSTATE<br>42804 | The host variable is of the date type and the data in the database is of another type and contains a value that cannot be interpreted as a date value.                                                                |
| -210<br>(ECPG_TIMESTAMP_FORMAT)  | SQLSTATE<br>42804 | The host variable is of the timestamp type and the data in the database is of another type and contains a value that cannot be interpreted as a timestamp value.                                                      |
| -211<br>(ECPG_CONVERT_BOOLEAN)   | SQLSTATE<br>42804 | The host variable is of the Boolean type, but the data in the database is neither 't' nor 'f'.                                                                                                                        |
| -212 (ECPG_EMPTY)                | SQLSTATE<br>YE000 | The statement sent to the SQL server was empty. (This usually does not occur in an embedded SQL program, so it may point to an internal error.)                                                                       |
| -213<br>(ECPG_MISSING_INDICATOR) | SQLSTATE<br>22002 | A null value was returned and no null indicator variable was provided.                                                                                                                                                |
| -214<br>(ECPG_NO_ARRAY)          | SQLSTATE<br>42804 | An ordinary variable was used in a place that requires an array.                                                                                                                                                      |
| -215<br>(ECPG_DATA_NOT_ARRAY)    | SQLSTATE<br>42804 | The database returned an ordinary variable in a place that requires array value.                                                                                                                                      |

| SQLCODE Value                             | SQLSTATE Value    | Description                                                                                                               |
|-------------------------------------------|-------------------|---------------------------------------------------------------------------------------------------------------------------|
| -216<br>(ECPG_ARRAY_INSERT)               | SQLSTATE<br>42804 | The value cannot be inserted into the array.                                                                              |
| -220<br>(ECPG_NO_CONN)                    | SQLSTATE<br>08003 | The program tried to access a connection that does not exist.                                                             |
| -221<br>(ECPG_NOT_CONN)                   | SQLSTATE<br>YE000 | The program tried to access a connection that does exist but is not open. (This is an internal error.)                    |
| -230<br>(ECPG_INVALID_STMT)               | SQLSTATE<br>26000 | The statement you are trying to use has not been prepared.                                                                |
| -239<br>(ECPG_INFORMIX_DUPLICATE_KEY)     | SQLSTATE<br>23505 | Duplicate key error, violation of unique constraint.                                                                      |
| -240<br>(ECPG_UNKNOWN_DESCRIPTOR)         | SQLSTATE<br>33000 | The specified descriptor was not found. The statement you are trying to use has not been prepared.                        |
| -241<br>(ECPG_INVALID_DESCRIPTOR_INDEX)   | SQLSTATE<br>07009 | The specified descriptor was out of range.                                                                                |
| -242<br>(ECPG_UNKNOWN_DESCRIPTOR_ITEM)    | SQLSTATE<br>YE000 | An invalid descriptor item was requested. (This is an internal error.)                                                    |
| -243<br>(ECPG_VAR_NOT_NUMERIC)            | SQLSTATE<br>07006 | During the execution of a dynamic statement, the database returned a numeric value and the host variable was not numeric. |
| -244<br>(ECPG_VAR_NOT_CHARACTER)          | SQLSTATE<br>07006 | During the execution of a dynamic statement, the database returned a non-numeric value and the host variable was numeric. |
| -284<br>(ECPG_INFORMIX_SUBSELECT_NOT_ONE) | SQLSTATE<br>21000 | The result of the subquery was not a single row.                                                                          |
| -400 (ECPG_PGSQL)                         | -                 | Some error caused by the SQL server. This message contains an error message from the SQL server.                          |
| -401 (ECPG_TRANS)                         | SQLSTATE<br>08007 | The SQL server notified that the transaction cannot be started, committed, or rolled back.                                |

| SQLCODE Value                         | SQLSTATE Value    | Description                                             |
|---------------------------------------|-------------------|---------------------------------------------------------|
| -402<br>(ECPG_CONNECT)                | SQLSTATE<br>08001 | The connection attempt to the database did not succeed. |
| -403<br>(ECPG_DUPLICATE_KEY)          | SQLSTATE<br>23505 | Duplicate key error, violation of unique constraint.    |
| -404<br>(ECPG_SUBSELECT_NOT_ONE)      | SQLSTATE<br>21000 | The result of the subquery was not a single row.        |
| -602<br>(ECPG_WARNING_UNKNOWN_PORTAL) | SQLSTATE<br>34000 | An invalid cursor name was specified.                   |
| -603<br>(ECPG_WARNING_IN_TRANSACTION) | SQLSTATE<br>25001 | A transaction is in progress.                           |
| -604<br>(ECPG_WARNING_NO_TRANSACTION) | SQLSTATE<br>25P01 | There is no active (in-progress) transaction.           |
| -605<br>(ECPG_WARNING_PORTAL_EXISTS)  | SQLSTATE<br>42P03 | An existing cursor name was specified.                  |

---

**⚠ CAUTION**

- The SQLSTATE codes 22002, 07001, 07002, 07006, 07009, 33000, 42601, 42804, 42P03, YE000, and YE001 are newly added to `ecpg` for embedded SQL statements. Other SQLSTATE codes are inherited from the kernel SQLSTATE codes.
  - If the value of **SQLSCODE** is **-400**, `ecpg` detects that the kernel server returns an error. The error code of the kernel SQLSTATE is used.
- 

## 5.8.11 Preprocessor Directives

This section describes the preprocessing instructions provided by `ECPG`. The preprocessing instructions are used to process program instructions for macro definition, file inclusion, and conditional compilation.

### 5.8.11.1 Including Files

To include an external file in an embedded SQL program, use the following statements:

```
EXEC SQL INCLUDE filename;
EXEC SQL INCLUDE <filename>;
EXEC SQL INCLUDE "filename";
```

 NOTE

- ECPG searches for files in the following sequence:
  1. Current directory
  2. /usr/local/include
  3. GaussDB Kernel directory, which is defined at build time
  4. /usr/include
- When **EXEC SQL INCLUDE** *filename* is used, only the current directory is searched.
- In each directory, ECPG will first look for the file name as given, and if not found will append .h to the file name and try again (unless the specified file name already has that suffix).
- The file name is case sensitive.

### 5.8.11.2 Directives: ifdef, ifndef, else, elif, and endif

ecpg provides ifdef, ifndef, else, elif, and endif conditional compilation instructions. During preprocessing, different parts of the program are compiled based on different conditions. When using the program, you need to add the EXEC SQL prefix keyword.

Example:

```
EXEC SQL ifndef TZVAR;
EXEC SQL SET TIMEZONE TO 'GMT';
EXEC SQL elif TZNAME;
EXEC SQL SET TIMEZONE TO TZNAME;
EXEC SQL else;
EXEC SQL SET TIMEZONE TO TZVAR;
EXEC SQL endif;
```

### 5.8.11.3 Directives: define and undef

Similar to the directive **#define** that is known from C, embedded SQL has a similar concept.

```
EXEC SQL DEFINE name;
EXEC SQL DEFINE name value;
EXEC SQL UNDEF name;
```

Example:

```
/* Define a name. */
EXEC SQL DEFINE HAVE_FEATURE;

/* Define constants. */
EXEC SQL DEFINE MYNUMBER 12;
EXEC SQL DEFINE MYSTRING 'abc';

/* Use undef to remove a previous definition. */
EXEC SQL UNDEF MYNUMBER;
```

You can also use the C versions **#define** and **#undef** in your embedded SQL program. The difference is where your defined values get evaluated. If you use **EXEC SQL DEFINE**, then ecpg evaluates the definitions and substitutes the values. In the following example, ecpg does the substitution and the compiler will never see any name or identifier **MYNUMBER**:

```
EXEC SQL DEFINE MYNUMBER 12;
...
EXEC SQL UPDATE Tbl SET col = MYNUMBER;
```

**NOTICE**

Note that you cannot use **#define** for a constant that you are going to use in an embedded SQL query because in this case the embedded SQL precompiler is not able to see this declaration.

## 5.8.12 Using Library Functions

- `ECPGdebug(int on, FILE *stream)`: If the first parameter of the function is not 0, the debug log function is enabled. The second parameter indicates the standard output stream of the log to be printed. Debug logs are executed on the standard output stream. The logs contain all input SQL statements and results from the GaussDB Kernel server.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
 ECPGdebug(1, stderr);
 EXEC SQL CONNECT postgres;
 EXEC SQL SET AUTOCOMMIT TO ON;
 EXEC SQL CREATE TABLE T1(a int);
 return (0);
}
```

- `ECPGget_PGconn(const char *connection_name)`: returns the database connection handle identified by the given name. If `connection_name` is set to **NULL**, the current connection handle is returned. If no connection handle can be identified, the function returns **NULL**.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
 ECPGdebug(1, stderr);
 EXEC SQL CONNECT TO postgres as con1;
 EXEC SQL SET AUTOCOMMIT TO ON;
 EXEC SQL DROP TABLE IF EXISTS T1;
 PGconn *conn;
 conn = ECPGget_PGconn("con1");
 printf("conn = %p\n", conn);
 conn = ECPGget_PGconn(NULL);
 printf("conn = %p\n", conn);
 EXEC SQL CREATE TABLE T1(a int);
 return (0);
}
```

- `ECPGtransactionStatus(const char *connection_name)`: returns the current transaction status of the `connection_name` connection. The possible return values are as follows:

```
PQTRANS_IDLE, /* connection idle */
PQTRANS_ACTIVE, /* command in progress */
PQTRANS_INTRANS, /* idle, within transaction block */
PQTRANS_INERROR, /* idle, within failed transaction */
PQTRANS_UNKNOWN /* cannot determine status */
```

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
 ECPGdebug(1, stderr);
 EXEC SQL CONNECT TO postgres as con1;
 EXEC SQL DROP TABLE IF EXISTS T1;
 int a = ECPGtransactionStatus("con1");
 printf("%d\n", a);
 EXEC SQL CREATE TABLE T1(a int);
 EXEC SQL COMMIT;
 return (0);
}
```

- `ECPGfree_auto_mem()`: releases all memory allocated for output host variables. This function is called (`return\exit`) when the program ends.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
 EXEC SQL BEGIN DECLARE SECTION;
 int *ip1=0;
 char **cp2=0;
 int *ipointer1=0;
 int *ipointer2=0;
 int colnum;
 EXEC SQL END DECLARE SECTION;
 int i;

 ECPGdebug(1, stderr);

 EXEC SQL WHENEVER SQLERROR DO sqlprint();
 EXEC SQL CONNECT TO REGRESSDB1;

 EXEC SQL SET DATESTYLE TO postgres;

 EXEC SQL CREATE TABLE test (a int, b text);
 EXEC SQL INSERT INTO test VALUES (1, 'one');
 EXEC SQL INSERT INTO test VALUES (2, 'two');
 EXEC SQL INSERT INTO test VALUES (NULL, 'three');
 EXEC SQL INSERT INTO test VALUES (NULL, NULL);

 EXEC SQL ALLOCATE DESCRIPTOR mydesc;
 EXEC SQL SELECT * INTO SQL DESCRIPTOR mydesc FROM test;
 EXEC SQL GET DESCRIPTOR mydesc :colnum=COUNT;
 EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :ip1=DATA, :ipointer1=INDICATOR;
 EXEC SQL GET DESCRIPTOR mydesc VALUE 2 :cp2=DATA, :ipointer2=INDICATOR;

 printf("Result (%d columns):\n", colnum);
 for (i=0; i < sqlca.sqlerrd[2]; ++i)
 {
 if (ipointer1[i]) printf("NULL, ");
 else printf("%d, ", ip1[i]);

 if (ipointer2[i]) printf("NULL, ");
 else printf("%s, ", cp2[i]);
 printf("\n");
 }
 ECPGfree_auto_mem();
 printf("\n");

 EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
```

```
EXEC SQL ROLLBACK;
EXEC SQL DISCONNECT;
return 0;
}
```

## 5.8.13 SQL Descriptor Area

An SQL descriptor area (SQLDA) is a more sophisticated method for processing the result of a SELECT, FETCH, or DESCRIBE statement. An SQLDA groups the data of one row of data together with metadata items into one data structure. ecpg provides two ways to use descriptor areas: named SQLDA and C-structure SQLDA.

### 5.8.13.1 Named SQLDA

A named SQLDA consists of a header and one or more item descriptor areas. The header contains information concerning the entire descriptor area, and each item descriptor area describes a column in the result row.

- Before using an SQLDA, you need to allocate it.  
EXEC SQL ALLOCATE DESCRIPTOR identifier;
- When you no longer need the SQLDA, deallocate it in time.  
EXEC SQL DEALLOCATE DESCRIPTOR identifier;
- To use a descriptor area, declare it using the INTO clause.  
EXEC SQL FETCH NEXT FROM mycursor INTO SQL DESCRIPTOR mydesc;

If the result set is empty, the descriptor area still contains the metadata from the query.

- For a prepared query that has not been executed, you can use DESCRIBE to obtain the metadata of its result set.

```
EXEC SQL BEGIN DECLARE SECTION;
char *sql_stmt = "SELECT * FROM table1";
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE stmt1 FROM :sql_stmt;
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
```

In DESCRIBE and FETCH statements, the INTO and USING keywords are used similarly: they produce a result set and metadata in a descriptor area.

- Retrieve the value in a descriptor area from the header and store it into a host variable.  
EXEC SQL GET DESCRIPTOR name :hostvar = field;
- Currently, only one header descriptor area **COUNT** is defined, which tells how many item descriptor areas exist (that is, how many columns are contained in the result). The host variable must be of the integer type. Retrieve a specific value from the item descriptor area.  
EXEC SQL GET DESCRIPTOR name VALUE num :hostvar = field;

**num** can be a character integer or a host variable that contains an integer. Possible data types are as follows:

- CARDINALITY (integer): number of rows in the result set
- DATA: actual data item (therefore, the data type of this field depends on the query)
- DATETIME\_INTERVAL\_CODE (integer): When **TYPE** is **9**, **DATETIME\_INTERVAL\_CODE** will have a value of **1** for DATE, **2** for TIME, **3** for TIMESTAMP, **4** for TIME WITH TIME ZONE, or **5** for TIMESTAMP WITH TIME ZONE.

- INDICATOR (integer): indicator (indicating a null value or a value truncation)
- LENGTH (integer): data length in characters
- NAME(string): column name
- OCTET\_LENGTH (integer): length of the character representation of the data in bytes
- PRECISION (integer): precision (for the numeric type)
- RETURNED\_LENGTH (integer): data length in characters
- RETURNED\_OCTET\_LENGTH (integer): length of the character representation of the data in bytes
- SCALE (integer): ratio (for the numeric type)
- TYPE (integer): numeric code of the data type of the column
- Retrieve the column value and store it in a host variable.  

```
EXEC SQL GET DESCRIPTOR mydesc VALUE num :hostvar = field
```

**num** can be a character integer or a host variable that contains an integer. Possible columns are as follows:

  - DATA
  - Actual data item (the data type of this column depends on the query)
  - NAME(string)
  - Column name
- Manually create a descriptor area to provide input parameters for a query or cursor.  

```
EXEC SQL SET DESCRIPTOR name VALUE numfield = :hostvar;
```
- Retrieve multiple rows of records in a FETCH statement and use a host variable of the array type to store data.  

```
EXEC SQL BEGIN DECLARE SECTION;
int id[5];
EXEC SQL END DECLARE SECTION;
EXEC SQL FETCH 5 FROM mycursor INTO SQL DESCRIPTOR mydesc;
EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :id = DATA;
```

### 5.8.13.2 C-Structure SQLDA

SQLDA is a C language structure used to store a query result set. A structure stores a record of a result set.

```
EXEC SQL include sqllda.h;
sqllda_t *mysqlda;
EXEC SQL FETCH 3 FROM mycursor INTO DESCRIPTOR mysqlda;
```

Note that the SQL keyword is omitted. The paragraphs about the use cases of the INTO and USING keywords in [Named SQLDA](#) also apply here. In a DESCRIBE statement, if the INTO keyword is used, the DESCRIPTOR keyword can be omitted.  

```
EXEC SQL DESCRIBE prepared_statement INTO mysqlda;
```

- Procedure
  - a. Prepare a query and declare a cursor for it.
  - b. Declare an SQLDA for the result row.
  - c. Declare SQLDA for input parameters, initialize parameters, and allocate memory.
  - d. Open a cursor with the input SQLDA.

- e. Fetch rows from the cursor and store them in the output SQLDA.
- f. Read the value from the output SQLDA to the host variable.
- g. Close the cursor.
- h. Deallocate the memory allocated to the SQLDA.
- There are three types of SQLDA data structures: `sqlda_t`, `sqlvar_t`, and `struct sqlname`.

a. `sqlda_t`

The definition of `sqlda_t` is as follows:

```
struct sqlda_struct
{
 char sqldaid[8];
 long sqldabc;
 short sqln;
 short sqld;
 struct sqlda_struct *desc_next;
 struct sqlvar_struct sqlvar[1];
};
typedef struct sqlda_struct sqlda_t;
```

The structure members are described as follows:

- **sqldaid**: contains a string "SQLDA".
- **sqldabc**: contains the size (in bytes) of the allocated space.
- **sqln**: contains the number of input parameters for a parameterized query in case it is passed into OPEN, DECLARE or EXECUTE statements using the USING keyword. When it is used as the output of a SELECT, EXECUTE, or FETCH statement, its value is the same as that of **sqld**.
- **sqld**: contains the number of fields in a result set.
- **desc\_next**: If the query returns more than one record, multiple linked SQLDA structures are returned, and **desc\_next** holds a pointer to the next SQLDA structure in the list.
- **sqlvar**: indicates the array of the columns in the result set.

b. `sqlvar_t`

The structure type `sqlvar_t` holds a column value and metadata (such as type and length). The definition of this type is as follows:

```
struct sqlvar_struct
{
 short sqltype;
 short sqllen;
 char *sqldata;
 short *sqlind;
 struct sqlname sqlname;
};
typedef struct sqlvar_struct sqlvar_t;
```

The structure members are described as follows:

- **sqltype**: contains the type identifier of the field.
- **sqllen**: contains the binary length of the field, for example, 4 bytes for `ECPGt_int`.

- **sqldata**: points to the data. For details about the data format, see [Type Mapping](#).
  - **sqlind**: points to a null indicator. The value **0** indicates not null, and the value **-1** indicates null.
  - **sqlname**: indicates the name of the field.
- c. struct sqlname
- A struct sqlname structure holds a column name. It is treated as a member of the sqlvar\_t structure. The definition of this type is as follows:
- ```
#define NAMEDATALEN 64
struct sqlname
{
    short    length;
    char     data[NAMEDATALEN];
};
```
- The structure members are described as follows:
- **length**: contains the length of the field name.
 - **data**: contains the actual field name.
- Use an SQLDA to retrieve a result set.
- The general procedure for retrieving a query result set through an SQLDA is as follows:
- a. Declare an sqlda_t structure to receive the result set.
 - b. Execute the **FETCH**, **EXECUTE**, or **DESCRIBE** command to process a query for which an SQLDA has been declared.
 - c. Check the number of records in the result set by looking at sqln, a member of the sqlda_t structure.
 - d. Fetch the values of each column from sqlvar[0], sqlvar[1], ..., members of the sqlda_t structure.
 - e. Go to next row (sqlda_t) by following the desc_next pointer, a member of the sqlda_t structure.
 - f. Repeat the preceding steps as required.

Example:

```
/* Declare an sqlda_t structure to receive the result set. */
sqlda_t *sqlda1;
/* Next, specify an SQLDA in a command. This is an example of the FETCH command. */
EXEC SQL FETCH NEXT FROM cur1 INTO DESCRIPTOR sqlda1;
/* Run a loop to retrieve rows along the linked list. */
sqlda_t *cur_sqlda;
for (cur_sqlda = sqlda1;
     cur_sqlda != NULL;
     cur_sqlda = cur_sqlda->desc_next)
{
    ...
}
/* Inside the loop, run another loop to retrieve the data of each column in the row (sqlvar_t). */
for (i = 0; i < cur_sqlda->sqln; i++)
{
    sqlvar_t v = cur_sqlda->sqlvar[i];
    char *sqldata = v.sqldata;
    short sqlllen = v.sqlllen;
    ...
}
/* To fetch the values of a column, check the value of the sqltype member of the sqlvar_t structure.
```

```
Then, switch to an appropriate method based on the column type to copy data from the sqlvar field to a host variable. */
char var_buf[1024];
switch (v.sqltype)
{
    case ECPGt_char:
        memset(&var_buf, 0, sizeof(var_buf));
        memcpy(&var_buf, sqldata, (sizeof(var_buf) <= sqlllen ? sizeof(var_buf) - 1 : sqlllen));
        break;

    case ECPGt_int:
        memcpy(&intval, sqldata, sqlllen);
        snprintf(var_buf, sizeof(var_buf), "%d", intval);
        break;

    ...
}
```

- Use an SQLDA to pass query parameters.

The general procedure for passing input parameters to a prepared query using an SQLDA is as follows:

- a. Create a prepared query (prepared statement).
- b. Declare an sqlda_t structure as an SQLDA.
- c. Allocate a memory area for the SQLDA.
- d. Set (copy) the input values in the allocated memory.
- e. Open a cursor declaring the SQLDA.

Example:

```
/* First, create a prepared statement. */
EXEC SQL BEGIN DECLARE SECTION;
char query[1024] = "SELECT d.oid, * FROM pg_database d, pg_stat_database s WHERE d.oid = s.datid AND (d.datname = ? OR d.oid = ?)";
EXEC SQL END DECLARE SECTION;
EXEC SQL PREPARE stmt1 FROM :query;

/* Allocate memory for an SQLDA and set the number of input parameters in the sqln member variable of the sqlda_t structure.
 * When the prepared query requires two or more input parameters, the application must allocate extra memory space. The space size is calculated as follows: (Number of parameters - 1) x sizeof(sqlvar_t).
 * The example here shows how to allocate memory space for two input parameters.
 */
sqlda_t *sqlda2;
sqlda2 = (sqlda_t *) malloc(sizeof(sqlda_t) + sizeof(sqlvar_t));
memset(sqlda2, 0, sizeof(sqlda_t) + sizeof(sqlvar_t));
sqlda2->sqln = 2; /* Number of input variables */
/* After memory allocation, store the parameter values into the sqlvar[] array. (This is same array used for retrieving column values when the SQLDA is receiving a result set.)
 * In this example, the input parameters are postgres (string type) and 1 (integer type). */
sqlda2->sqlvar[0].sqltype = ECPGt_char;
sqlda2->sqlvar[0].sqldata = "postgres";
sqlda2->sqlvar[0].sqlllen = 8;
int intval = 1;
sqlda2->sqlvar[1].sqltype = ECPGt_int;
sqlda2->sqlvar[1].sqldata = (char *) &intval;
sqlda2->sqlvar[1].sqlllen = sizeof(intval);
/* Input parameters are passed to the prepared statement by opening a cursor and declaring the SQLDA that has been created. */
EXEC SQL OPEN cur1 USING DESCRIPTOR sqlda2;
/* Finally, the allocated memory must be explicitly released after you use the input SQLDA, which is different from the SQLDA used to receive query results. */
free(sqlda2);
```

5.8.14 Examples

ecpg Example Code

```
#include <locale.h>
#include <string.h>
#include <stdlib.h>

exec sql whenever sqlerror sqlprint;
exec sql include sqlca;

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    char *temp_str = (char *)malloc(11);
EXEC SQL END DECLARE SECTION;

    ECPGdebug(1, stderr);

    exec sql connect to postgres;

    /* Enable the automatic commit function. You do not need to manually commit the exec sql command. */
    exec sql set autocommit = on;
    exec sql drop table if exists test_t;
    /* Create a table and insert data. */
    exec sql create table test_t(f float, i int, a int[10], mstr char(10));
    exec sql insert into test_t(f, i, a, mstr) values(1.01,1,{0,1,2,3,4,5,6,7,8,9}', 'China');

    /* Disable the automatic commit function. The following SQL statements for inserting data must be
    manually committed: */
    exec sql set autocommit = off;
    exec sql insert into test_t(f, i, a, mstr) values(2.01,2,{0,1,2,3,4,5,6,7,8,9}', 'USA');
    exec sql commit;

    exec sql insert into test_t(f, i, a, mstr) values(3.01,3,{0,1,2,3,4,5,6,7,8,9}', 'AUS');
    exec sql insert into test_t(f, i, a, mstr) values(4.01,4,{0,1,2,3,4,5,6,7,8,9}', 'JAP');
    exec sql commit;

EXEC SQL BEGIN DECLARE SECTION;
    int a[10] = {9,8,7,6,5,4,3,2,1,0};
    int id = 6;
EXEC SQL END DECLARE SECTION;

    /* Fetch data from the host variable and insert the data into the table. The type of the host variable is
    the same as that defined in the table. */
    strcpy(temp_str, "RUS");
    exec sql insert into test_t(f, i, a, mstr) values(5.01,5,:a,:temp_str);
    exec sql commit;

    exec sql set autocommit = on;
    exec sql begin;
    exec sql insert into test_t(f, i, a, mstr) values(6.01,:id,:a,'SIG');
    exec sql commit;
    exec sql set autocommit = off;

exec sql begin declare section;
    float ff;
    char tmp_text[25] = "klmnopqrst";
exec sql end declare section;

    exec sql set autocommit = on;
    exec sql begin work;

    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    /* Example of a conditional query statement */
    exec sql select f, mstr into :ff,:tmp_text from test_t where f > (select f from test_t where i = 4 or i < 0)
    order by a limit 1;
```

```
printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

exec sql select f, mstr into :ff,:tmp_text from test_t where mstr = 'JAP' order by i;
printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

exec sql select f, mstr into :ff,:tmp_text from test_t order by i DESC limit 1;
printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

exec sql select f, mstr into :ff,:tmp_text from test_t order by mstr limit 1;
printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

exec sql select count(f), a into :ff,:tmp_text from test_t where i > 2 group by a limit 1;
printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

exec sql select count(f), a into :ff,:tmp_text from test_t where i > 3 group by a order by a limit 1;
printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

exec sql select sum(f), a into :ff,:tmp_text from test_t where i > 2 group by a order by a limit 1;
printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

exec sql select distinct a into :tmp_text from test_t order by a limit 1;

exec sql drop table test_t;

exec sql commit;
/* Release the connection and release the memory allocated to the host variable. */
exec sql disconnect;
free(temp_str);

return 0;
}
```

Example Code of the pgtypes Library Function

Example 1: Use library functions to perform different operations on time and date types. For details, see [Using Library Functions](#).

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>
#include <pgtypes_date.h>
#include <pgtypes_timestamp.h>

char *dates[] = { "19990108foobar",
                 "19990108 foobar",
                 "1999-01-08 foobar",
                 "January 8, 1999",
                 "1999-01-08",
                 "1/8/1999",
                 "1/18/1999",
                 "01/02/03",
                 "1999-Jan-08",
                 "Jan-08-1999",
                 "08-Jan-1999",
                 "99-Jan-08",
                 "08-Jan-99",
                 "08-Jan-06",
                 "Jan-08-99",
                 "19990108",
                 "990108",
                 "1999.008",
                 "J2451187",
                 "January 8, 99 BC",
                 NULL
};

/* The value cannot conflict with the value of times of libc. */
static char *times[] = { "0:04",
```

```
        "1:59 PDT",
        "13:24:40 -8:00",
        "13:24:40.495+3",
        NULL
    };

char *intervals[] = { "1 minute",
    "1 12:59:10",
    "2 day 12 hour 59 minute 10 second",
    "1 days 12 hrs 59 mins 10 secs",
    "1 days 1 hours 1 minutes 1 seconds",
    "1 year 59 mins",
    "1 year 59 mins foobar",
    NULL
};

int main(void)
{
    exec sql begin declare section;
        date date1;
        timestamp ts1, ts2;
        char *text;
        interval *i1;
        date *dc;
    exec sql end declare section;

    int i, j;
    char *endptr;

    ECPGdebug(1, stderr);

    /* Parse a timestamp from its textual representation and convert the date into a character string. */
    ts1 = PGTYPEStimestamp_from_asc("2003-12-04 17:34:29", NULL);
    text = PGTYPEStimestamp_to_asc(ts1);

    printf("timestamp: %s\n", text);
    free(text);

    /* Extract the date part from the timestamp. */
    date1 = PGTYPEStimestamp_to_date(ts1);
    dc = PGTYPEStimestamp_to_date_new(date1);
    *dc = date1;
    /* Return the textual representation of a date variable. */
    text = PGTYPEStimestamp_to_date_to_asc(*dc);
    printf("Date of timestamp: %s\n", text);
    free(text);
    PGTYPEStimestamp_to_date_free(dc);

    for (i = 0; dates[i]; i++)
    {
        bool err = false;
        /* Parse a date from the textual representation of the date. */
        date1 = PGTYPEStimestamp_to_date_from_asc(dates[i], &endptr);
        if (date1 == INT_MIN) {
            err = true;
        }
        /* Return the textual representation of a date variable. */
        text = PGTYPEStimestamp_to_date_to_asc(date1);
        printf("Date[%d]: %s (%c - %c)\n",
            i, err ? "-" : text,
            endptr ? 'N' : 'Y',
            err ? 'T' : 'F');
        free(text);
        if (!err)
        {
            for (j = 0; times[j]; j++)
            {
                int length = strlen(dates[i]) + 1 + strlen(times[j]) + 1;
                char* t = (char *)malloc(length);
```

```
    sprintf(t, "%s %s", dates[i], times[j]);
    /* Parse a timestamp from its textual representation and convert the date into a character string.
*/
    ts1 = PGTYPEStimestamp_from_asc(t, NULL);
    text = PGTYPEStimestamp_to_asc(ts1);
    if (i != 19 || j != 3)
        printf("TS[%d,%d]: %s\n", i, j, errno ? "-" : text);
    free(text);
    free(t);
    }
}

/* Parse a timestamp from its textual representation. */
ts1 = PGTYPEStimestamp_from_asc("2004-04-04 23:23:23", NULL);

for (i = 0; intervals[i]; i++)
{
    interval *ic;
    /* Parse an interval from its textual representation. */
    i1 = PGTYPEStimestamp_from_asc(intervals[i], &endptr);
    if (*endptr)
        printf("endptr set to %s\n", endptr);
    if (!i1)
    {
        printf("Error parsing interval %d\n", i);
        continue;
    }
    /* Add an interval variable to the timestamp variable. */
    j = PGTYPEStimestamp_add_interval(&ts1, i1, &ts2);
    if (j < 0)
        continue;
    /* Convert a variable of the interval type to the text format. */
    text = PGTYPEStimestamp_to_asc(i1);
    printf("interval[%d]: %s\n", i, text ? text : "-");
    free(text);

    /* Return a pointer to an allocated interval variable. */
    ic = PGTYPEStimestamp_new();
    /* Copy a variable of the interval type. */
    PGTYPEStimestamp_copy(i1, ic);
    /* Convert a variable of the interval type to the text format. */
    text = PGTYPEStimestamp_to_asc(i1);
    printf("interval_copy[%d]: %s\n", i, text ? text : "-");
    free(text);
    /* Release the memory that has been allocated to an interval variable. */
    PGTYPEStimestamp_free(ic);
    PGTYPEStimestamp_free(i1);
}

return (0);
}
```

Example 2: Use the pgtypes library function to perform different operations on the numeric type.

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_numeric.h>
#include <pgtypes_error.h>
#include <decimal.h>

char* nums[] = { "2E394", "-2", ".794", "3.44", "592.49E21", "-32.84e4",
    "2E-394", ".1E-2", "+.0", "-592.49E-07", "+32.84e-4",
    ".500001", "-.5000001",
    "1234567890123456789012345678.91", /* A 30-bit number should be converted into a decimal
number.*/
    "1234567890123456789012345678.921", /* A 31-bit number cannot be converted into a
decimal number. */
    "not a number",
```

```
        NULL
    };

static void check_errno(void);

int main(void)
{
    char *text="error\n";
    char *endptr;
    numeric *num, *nin;
    decimal *dec;
    long l;
    int i, j, k, q, r, count = 0;
    double d;
    numeric **numarr = (numeric **) calloc(1, sizeof(numeric));

    ECPGdebug(1, stderr);

    for (i = 0; nums[i]; i++)
    {
        /* Return a pointer to a string allocated by malloc that contains the string representation of the
        numeric type nums[i]. */
        num = PGTYPEStoasc(nums[i], &endptr);
        if (!num) check_errno();
        if (endptr != NULL)
        {
            printf("endptr of %d is not NULL\n", i);
            if (*endptr != '\0')
                printf("**endptr of %d is not \\0\n", i);
        }
        if (!num) continue;

        numarr = (numeric **)realloc(numarr, sizeof(numeric *) * (count + 1));
        numarr[count++] = num;

        /* Return a pointer to a string allocated by malloc that contains the string representation of the
        numeric type num. */
        text = PGTYPEStoasc(num, -1);
        if (!text) check_errno();
        printf("num[%d,1]: %s\n", i, text); free(text);
        text = PGTYPEStoasc(num, 0);
        if (!text) check_errno();
        printf("num[%d,2]: %s\n", i, text); free(text);
        text = PGTYPEStoasc(num, 1);
        if (!text) check_errno();
        printf("num[%d,3]: %s\n", i, text); free(text);
        text = PGTYPEStoasc(num, 2);
        if (!text) check_errno();
        printf("num[%d,4]: %s\n", i, text); free(text);

        /* Request a pointer to a newly allocated numeric variable. */
        nin = PGTYPEStoasc(num, 2);
        if (!nin) check_errno();
        printf("num[%d,5]: %s\n", i, nin); free(nin);

        /* Convert a numeric variable into a long int variable. */
        r = PGTYPEStoasc(num, &l);
        if (r) check_errno();
        printf("num[%d,6]: %ld (r: %d)\n", i, l, r);
        if (r == 0)
        {
            /* Convert a long int variable into a numeric variable. */
            r = PGTYPEStoasc(l, nin);
            if (r) check_errno();
        }
        /* Return a pointer to a string allocated by malloc that contains the string representation of the
        numeric type nin. */
        text = PGTYPEStoasc(nin, 2);
        /* Compare two numeric variables. */
    }
}
```

```
    q = PGTYPENumeric_cmp(num, nin);
    printf("num[%d,7]: %s (r: %d - cmp: %d)\n", i, text, r, q);
    free(text);
}

/* Convert a numeric variable into an int variable. */
r = PGTYPENumeric_to_int(num, &k);
if (r) check_errno();
printf("num[%d,8]: %d (r: %d)\n", i, r?0:k, r);
if (r == 0)
{
    /* Convert an int variable into a numeric variable. */
    r = PGTYPENumeric_from_int(k, nin);
    if (r) check_errno();
    /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nin. */
    text = PGTYPENumeric_to_asc(nin, 2);
    q = PGTYPENumeric_cmp(num, nin);
    printf("num[%d,9]: %s (r: %d - cmp: %d)\n", i, text, r, q);
    free(text);
}

if (i != 6)
{
    /* Convert a variable of the numeric type to the double-precision type. */
    r = PGTYPENumeric_to_double(num, &d);
    if (r) check_errno();
    printf("num[%d,10]: %g (r: %d)\n", i, r?0.0:d, r);
}

/* Request a pointer to a newly allocated numeric variable. */
dec = PGTYPEDecimal_new();
/* Convert a decimal variable into a numeric variable. */
r = PGTYPENumeric_to_decimal(num, dec);
if (r) check_errno();
printf("num[%d,11]: - (r: %d)\n", i, r);
if (r == 0)
{
    /* Convert a decimal variable into a numeric variable. */
    r = PGTYPENumeric_from_decimal(dec, nin);
    if (r) check_errno();
    /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nin. */
    text = PGTYPENumeric_to_asc(nin, 2);
    /* Compare two numeric variables. */
    q = PGTYPENumeric_cmp(num, nin);
    printf("num[%d,12]: %s (r: %d - cmp: %d)\n", i, text, r, q);
    free(text);
}

/* Release the memory allocated to the numeric variable. */
PGTYPEDecimal_free(dec);
PGTYPENumeric_free(nin);
printf("\n");
}

for (i = 0; i < count; i++)
{
    for (j = 0; j < count; j++)
    {
        /* Request a pointer to a newly allocated numeric variable. */
        numeric* a = PGTYPENumeric_new();
        numeric* s = PGTYPENumeric_new();
        numeric* m = PGTYPENumeric_new();
        numeric* d = PGTYPENumeric_new();
        /* Add two numeric variables to the third numeric variable. */
        r = PGTYPENumeric_add(numarr[i], numarr[j], a);
        if (r)
        {
```

```
        check_errno();
        printf("r: %d\n", r);
    }
    else
    {
        /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type a. */
        text = PGTYPEPNumeric_to_asc(a, 10);
        printf("num[a,%d,%d]: %s\n", i, j, text);
        free(text);
    }
    /* Subtract two numeric variables and returns the result to the third numeric variable. */
    r = PGTYPEPNumeric_sub(numarr[i], numarr[j], s);
    if (r)
    {
        check_errno();
        printf("r: %d\n", r);
    }
    else
    {
        /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type s. */
        text = PGTYPEPNumeric_to_asc(s, 10);
        printf("num[s,%d,%d]: %s\n", i, j, text);
        free(text);
    }
    /* Multiply two numeric variables and returns the result to the third numeric variable. */
    r = PGTYPEPNumeric_mul(numarr[i], numarr[j], m);
    if (r)
    {
        check_errno();
        printf("r: %d\n", r);
    }
    else
    {
        /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type m. */
        text = PGTYPEPNumeric_to_asc(m, 10);
        printf("num[m,%d,%d]: %s\n", i, j, text);
        free(text);
    }
    /* Divide two numeric variables and returns the result to the third numeric variable. */
    r = PGTYPEPNumeric_div(numarr[i], numarr[j], d);
    if (r)
    {
        check_errno();
        printf("r: %d\n", r);
    }
    else
    {
        /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type d. */
        text = PGTYPEPNumeric_to_asc(d, 10);
        printf("num[d,%d,%d]: %s\n", i, j, text);
        free(text);
    }

    /* Release the memory allocated to the numeric variable. */
    PGTYPEPNumeric_free(a);
    PGTYPEPNumeric_free(s);
    PGTYPEPNumeric_free(m);
    PGTYPEPNumeric_free(d);
}
}

for (i = 0; i < count; i++)
{
    /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type numarr[i]. */
```

```
text = PGTYPESto_asc(numarr[i], -1);
printf("%d: %s\n", i, text);
free(text);
/* Free memory. */
PGTYPESto_free(numarr[i]);
}
free(numarr);

return (0);
}

/* Error handling */
static void
check_errno(void)
{
    switch(errno)
    {
        case 0:
            printf("(no errno set) - ");
            break;
        case PGTYPESto_OVERFLOW:
            printf("(errno == PGTYPESto_OVERFLOW) - ");
            break;
        case PGTYPESto_UNDERFLOW:
            printf("(errno == PGTYPESto_UNDERFLOW) - ");
            break;
        case PGTYPESto_BAD_NUMERIC:
            printf("(errno == PGTYPESto_BAD_NUMERIC) - ");
            break;
        case PGTYPESto_DIVIDE_ZERO:
            printf("(errno == PGTYPESto_DIVIDE_ZERO) - ");
            break;
        default:
            printf("(unknown errno (%d))\n", errno);
            printf("(libc: (%s) ", strerror(errno));
            break;
    }
}
```

5.8.15 ecpg and Pro*C Compatibility Comparison

ecpg, provided by GaussDB, is an embedded SQL preprocessor for C programs. It differs from the Pro*C preprocessor of the ORA-compatible database in the behavior and semantics of compiling and executing commands, syntax, and embedded statements.

Comparison between ecpg and Pro*C:

- Currently, ecpg does not support EXEC SQL CONTEXT ALLOCATE, EXEC SQL CONTEXT USE, and EXEC SQL Context FREE.

NOTE

Currently, ecpg does not support context allocation, use, and release. ecpg has an independent memory management mechanism. In multi-thread mode, ecpg independently establishes connections, executes SQL statements, and releases related resources in each thread. This usage mode is the same as the processing logic for each thread to allocate and release contexts in Pro*C multi-thread mode.

- Currently, ecpg does not support EXEC SQL COMMIT WORK RELEASE.

 **NOTE**

In ecpg, after the COMMIT statement is executed, the **RELEASE** option is not available. You need to execute the EXEC SQL DISCONNECT and EXEC SQL CLOSE statements to release related resources. In Pro*C, EXEC SQL COMMIT has the **RELEASE** option, which is used to release all resource information such as connections and cursors held by the program.

- Currently, ecpg does not support EXEC SQL ENABLE THREAD.

 **NOTE**

To enable macro definition in the ecpg compilation options, you need to define ENABLE_THREAD_SAFETY in the .pgc file of the main function.

5.8.16 ECPG API Reference

The ECPG API reference describes the data type-related interfaces provided by the pgtypes library for users to use in embedded SQL-C programs. The pgtypes library maps SQL data types to C data types and provides some interfaces to implement basic functions and calculations.

5.8.16.1 Interval Type

[Table 5-90](#) lists the common APIs for interval data provided by ecpg.

Table 5-90 Common interval type APIs

API	Description	Remarks
interval* PGTYPEInterval_new(void)	Returns a pointer to an allocated interval variable.	This function creates an interval variable on the heap. The return value is of the interval* type.
void PGTYPEInterval_free(interval* inval)	Releases the memory that has been allocated to an interval variable.	This function releases the memory allocated to the interval* variable created by the PGTYPEInterval_new function.
interval* PGTYPEInterval_from_asc(char* str, char** endptr)	Parses an interval from its textual representation.	This function parses the input string str and returns a pointer to an allocated interval variable. Currently, ecpg parses the entire character string and does not support storing the address of the first invalid character in *endptr . endptr can be set to NULL .

API	Description	Remarks
char* PGTYPEInterval_to_asc(interval* span)	Converts a variable of the interval type into its textual representation.	This function converts the interval variable to which the span points into a char*.
int PGTYPEInterval_copy(interval* intvlsrc, interval* intvldest)	Copies a variable of the interval type.	This function copies the interval variable that intvlsrc points to into the variable that intvldest points to.

Examples

For details, see [Examples](#).

5.8.16.2 Numeric Type

[Table 5-91](#) lists the common APIs for numeric (numeric or decimal) data provided by ecpg.

Table 5-91 Common numeric type APIs

API	Description	Remarks
numeric* PGTYPEnumeric_new(void)	Requests a pointer to a newly allocated numeric variable.	This function creates a numeric variable on the heap. The return value is of the numeric* type.
decimal* PGTYPEdecimal_new(void)	Requests a pointer to a newly allocated decimal variable.	This function creates a decimal variable on the heap. The return value is of the decimal* type.
void PGTYPEnumeric_free(numeric* var)	Frees the memory of a variable of the numeric type.	This function frees a numeric* variable created by using the PGTYPEnumeric_new function.
void PGTYPEdecimal_free(decimal*)	Frees the memory of a variable of the decimal type.	This function frees a decimal* variable created by using the PGTYPEdecimal_new function.

API	Description	Remarks
numeric* PGTYPEsnumeric_from_asc(char* str, char** endptr)	Converts a string to the numeric type.	For example, the valid formats are -2, .794, +3.44, 592.49E07, or -32.84e-4. If the value could be parsed successfully, a valid pointer is returned; otherwise, a null pointer is returned. ecpg only parses complete character strings. Currently, ecpg does not support storing the address of the first invalid character in *endptr. However, endptr can be set to null.
char* PGTYPEsnumeric_to_asc(numeric* num, int dscale)	Returns a pointer to a string allocated by malloc that contains the string representation of the numeric type num.	The numeric value will be printed using dscale decimal places and will be rounded if necessary.
int PGTYPEsnumeric_add(numeric* var1, numeric* var2, numeric* result)	Adds two numeric variables to the third numeric variable.	This function adds the variables var1 and var2 to the result variable result. The function returns 0 on success and -1 in case of error.
int PGTYPEsnumeric_sub(numeric* var1, numeric* var2, numeric* result)	Subtracts two numeric variables and returns the result to the third numeric variable.	This function subtracts var2 from var1. The result of this operation is stored in the variable result. The function returns 0 on success and -1 in case of error.
int PGTYPEsnumeric_mul(numeric* var1, numeric* var2, numeric* result)	Multiplies two numeric variables and returns the result to the third numeric variable.	This function multiplies variables var1 and var2. The result of this operation is stored in the variable result. The function returns 0 on success and -1 in case of error.

API	Description	Remarks
int PGTYPESnumeric_div(nu meric* var1, numeric* var2, numeric* result)	Divides two numeric variables and returns the result to the third numeric variable.	This function divides variable <i>var1</i> by variable <i>var2</i> . The result of this operation is stored in the variable <i>result</i> . The function returns 0 on success and -1 in case of error.
int PGTYPESnumeric_cmp(n umeric* var1, numeric* var2)	Compares two numeric variables.	This function compares two numeric variables. If an error occurs, <i>INT_MAX</i> is returned. On success, this function returns one of the following three possible results: <ul style="list-style-type: none"> • If <i>var1</i> is greater than <i>var2</i>, 1 is returned. • If <i>var1</i> is smaller than <i>var2</i>, -1 is returned. • If <i>var1</i> and <i>var2</i> are equal, 0 is returned.
int PGTYPESnumeric_from_i nt(signed int int_val, numeric* var)	Converts an int variable into a numeric variable.	This function accepts a signed int variable and stores it in the numeric variable <i>var</i> . The function returns 0 on success and -1 in case of failure.
int PGTYPESnumeric_from_l ong(signed long int long_val, numeric* var)	Converts a long int variable into a numeric variable.	This function accepts a long int variable and stores it in the numeric variable <i>var</i> . The function returns 0 on success and -1 in case of failure.
int PGTYPESnumeric_copy(n umeric* src, numeric* dst)	Copies a numeric variable to another one.	This function copies the value of the variable that src points to into the variable that dst points to. The function returns 0 on success and -1 in case of error.

API	Description	Remarks
int PGTYPESnumeric_from_double(double d, numeric* dst)	Converts a double-precision variable into a numeric variable.	This function accepts a double-precision variable and stores the result in the variable that dst points to. The function returns 0 on success and -1 in case of error.
int PGTYPESnumeric_to_double(numeric* nv, double* dp)	Converts a numeric variable into a double-precision variable.	This function converts the numeric value in the variable that nv points to into a double-precision variable that dp points to. The function returns 0 on success and -1 in case of error (or overflow). When overflow occurs, the global variable errno is additionally set to PGTYPES_NUM_OVERFLOW .
int PGTYPESnumeric_to_int(numeric* nv, int* ip)	Converts a numeric variable into an int variable.	This function converts the numeric value in the variable that nv points to into an int variable that ip points to. The function returns 0 on success and -1 in case of error (or overflow). When overflow occurs, the global variable errno is additionally set to PGTYPES_NUM_OVERFLOW .
int PGTYPESnumeric_to_long(numeric* nv, long* ip)	Converts a numeric variable into a long int variable.	This function converts the numeric value in the variable that nv points to into a long int variable that ip points to. The function returns 0 on success and -1 in case of error (or overflow). When overflow occurs, the global variable errno is additionally set to PGTYPES_NUM_OVERFLOW .

API	Description	Remarks
int PGTYPESnumeric_to_decimal(numeric* src, decimal* dst)	Converts a numeric variable into a decimal variable.	This function converts the numeric value in the variable that src points to into a decimal variable that dst points to. The function returns 0 on success and -1 in case of error (or overflow). When overflow occurs, the global variable errno is additionally set to PGTYPES_NUM_OVERFLOW .
int PGTYPESnumeric_from_decimal(decimal* src, numeric* dst)	Converts a decimal variable into a numeric variable.	This function converts the decimal value in the variable that src points to into a numeric variable that dst points to. The function returns 0 on success and -1 in case of error (or overflow).

Examples

For details, see [Examples](#).

5.8.16.3 Date Type

[Table 5-92](#) lists the common date type APIs provided by ecpg.

Table 5-92 Common date type APIs

API	Description	Remarks
date* PGTYPESdate_new(void)	Returns a pointer to an allocated date variable.	This function creates a date variable on the heap. The return value is of the date* type.
void PGTYPESdate_free(date*)	Releases the memory that has been allocated to a date variable.	Frees a date* variable created using the PGTYPESdate_free function.

API	Description	Remarks
<p>date PGTYPESdate_from_asc(char* str, char** endptr)</p>	<p>Parses a date from its textual representation.</p>	<p>This function accepts a C string str and a pointer to a C string endptr. ecpg converts the date expressed in text into a character string. Currently, ECPG does not support storing the address of the first invalid character in *endptr. However, endptr can be set to null.</p> <p>Note that the function always assumes MDY-formatted dates and there is currently no variable to change that within ecpg.</p>
<p>char* PGTYPESdate_to_asc(date dDate)</p>	<p>Returns the textual representation of a date variable.</p>	<p>This function accepts the date dDate as its unique parameter. It will output the date in the YYYY-MM-DD format.</p>
<p>date PGTYPESdate_from_timestamp(timestamp dt)</p>	<p>Extracts the date part from a timestamp.</p>	<p>This function accepts a timestamp as its unique parameter and returns the extracted date part from the timestamp.</p>
<p>void PGTYPESdate_julmdy(date jd, int* mdy)</p>	<p>Extracts the values of day, month, and year from a variable of the date type.</p>	<p>This function accepts the date jd and a pointer to an array of three integer values mdy. The variable name indicates the sequence. mdy[0] indicates the month, mdy[1] indicates the day, and mdy[2] indicates the year.</p>

API	Description	Remarks
void PGTYPEdate_mdyjul(int * mdy, date* jdate)	Creates a date using the specified integer value.	This function accepts an array consisting of three integers (mdy) as its first parameter. The three integers are used to indicate the day, month, and year, respectively. The second parameter is a pointer to a variable of the date type, which is used to store the result of the operation.
int PGTYPEdate_dayofweek(date d)	Returns a number indicating the day of a week for a date value.	This function accepts the date variable d as its unique parameter and returns an integer indicating the day of the week.
void PGTYPEdate_today(date * d)	Returns the current date.	This function accepts a pointer to a date variable <i>d</i> and sets the parameter to the current date. <ul style="list-style-type: none"> ● 0: Sunday ● 1: Monday ● 2: Tuesday ● 3: Wednesday ● 4: Thursday ● 5: Friday ● 6: Saturday

API	Description	Remarks
int PGTYPEStime_defmt_asc(date* d, const char* fmt, char* str)	Converts a C char* string to a value of the date type using a format mask.	This function accepts a pointer d pointing to a date value for storing the operation result, a format mask fmt for parsing the date, and a C char* string str containing the textual representation of the date. The textual representation is expected to match the format mask. However, you do not need to have a 1:1 mapping of the string to the format mask. The function only analyzes the sequential order and looks for the literals "yy" or "yyyy" that indicate the position of the year, "mm" indicating the position of the month and "dd" indicating the position of the day. Example of valid input (fmt, str): yy/mm/dd 1994, February 3rd

API	Description	Remarks
int PGTYPEdate_fmt_asc(t dDate, const char* fmtstring, char* outbuf)	Converts a variable of the date type into its textual representation using a format mask.	This function accepts the date dDate to be converted, the format mask fmtstring , and the string outbuf of the textual representation of the date to be saved. The function returns 0 on success and a negative value in case of error. Example of valid input: 112359 //mmddy 59/11/23 //yy/mm/dd Nov.23,1959 //mmm.dd,yyyy Mon,Nov.23,1959 // ddd,mmm.dd,yyyy Format: <ul style="list-style-type: none"> • dd: indicates the day of a month. • mm: indicates the month of a year. • yy: indicates a two-digit year. • yyyy: indicates a four-digit year. • ddd: indicates the day of a week. • mmm: indicates the month.

Examples

For details, see [Examples](#).

5.8.16.4 Timestamp Type

[Table 5-93](#) lists the common APIs for timestamp data provided by ecpg.

Table 5-93 Common timestamp type APIs

API	Description	Remarks
timestamp PGTYPEStimestamp_from_asc(char *str, char **endptr)	Parses a timestamp from its textual representation into a timestamp variable.	This function accepts a string str to parse and a pointer to a C char* endptr . It returns the parsed timestamp on success and PGTYPEStimestamp in case of error. In addition, errno is set to PGTYPEStimestamp when an error occurs. The valid input of PGTYPEStimestamp_from_asc is as follows: 1999-01-08 04:05:06 January 8 04:05:06 1999 PST 1999-Jan-08 04:05:06.789-8 1999-01-08 04:05:06.789 (time zone specifier ignored) J2451187 04:05-08:00 1999-01-08 04:05:00 (time zone specifier ignored)
char *PGTYPEStimestamp_to_asc(timestamp tstamp)	Converts a date to a C char* string.	This function accepts the timestamp tstamp as its unique parameter and returns an allocated string containing the textual representation of the timestamp. The result must be released using PGTYPEStimestamp_free() .
void PGTYPEStimestamp_current(timestamp *ts)	Returns the current timestamp.	This function obtains the current timestamp and saves it to the timestamp variable that ts points to.

API	Description	Remarks
<p>int PGTYPEtimestamp_fmt_asc(timestamp *ts, char *output, int str_len, char *fmtstr)</p>	<p>Converts a timestamp variable into a char* using a format mask.</p>	<p>This function accepts a pointer pointing to the timestamp ts to be converted, a pointer pointing to the output buffer output, the maximum length allocated to the output buffer str_len, and the format mask fmtstr for conversion as its parameters.</p> <p>The function returns 0 on success and a negative value in case of error.</p>
<p>int PGTYPEtimestamp_sub(timestamp *ts1, timestamp *ts2, interval *iv)</p>	<p>Subtracts a timestamp from another timestamp and saves the result in an interval variable.</p>	<p>This function subtracts the timestamp variable that ts2 points to from the timestamp variable that ts1 points to, and stores the result in the interval variable that iv points to.</p> <p>The function returns 0 on success and a negative value in case of error.</p>
<p>int PGTYPEtimestamp_defmt_asc(char *str, char *fmt, timestamp *d)</p>	<p>Parses the timestamp value from its textual representation using a format mask.</p>	<p>This function accepts a timestamp textual representation placed in the variable str and a format mask placed in the variable fmt that will be used for conversion. The result will be stored in the variable that d points to.</p> <p>If the format mask fmt is null, the function rolls back to use the default format mask %Y- %m- %d %H: %M: %S.</p>

API	Description	Remarks
int PGTYPEStimestamp_add_ interval(timestamp *tin, interval *span, timestamp *tout)	Adds an interval variable to a timestamp variable.	This function accepts a pointer tin pointing to the timestamp variable and a pointer span pointing to the interval variable. It adds the interval to the timestamp and then saves the result timestamp in the variable that tout points to. The function returns 0 on success and a negative value in case of error.
int PGTYPEStimestamp_sub_ interval(timestamp* tin, interval* span, timestamp* tout)	Subtracts an interval variable from a timestamp variable.	This function subtracts the interval variable that span points to from the timestamp variable that tin points to, and then saves the result in the variable that tout points to. The function returns 0 on success and a negative value in case of error.

Examples

For details, see [Examples](#).

5.9 Development Based on the Go Driver

5.9.1 Setting Up the Go Driver Environment

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-94](#).

Table 5-94 Driver package download list

Version	Download Address
V2.0-3.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

- Verifying the software package integrity on a Linux server:
 - a. Upload the software package and verification package to the same directory on the VM.
 - b. Run the following command to verify the integrity of the software package:

```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

If **OK** is displayed in the command output, the verification is successful.

```
GaussDB_driver.zip: OK
```
- Verifying the software package integrity on a Windows server:
 - a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
 - b. Run the following command to obtain the hash value of the driver package:

```
certutil -hashfile {local_directory_of_the_driver_package}\{driver_package_name} sha256
```

 - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
 - Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **GaussDB_driver.zip**.Example: **certutil -hashfile C:\Users\GaussDB_driver.zip sha256**
 - c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-94**.
 - If they are consistent, the verification is successful.
 - If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Environment Class

- **Configure the Go environment.**

You need to configure the following parameters in the environment variables:

 - **GO111MODULE**: Set **GO111MODULE** to **on** when installing the Go driver by importing a file online. If you do not want to reconstruct the **go mod** project, set **GO111MODULE** to **off** and manually download the dependency package. The dependency package must be at the same level as the driver root directory and service code.

- **GOPROXY**: When importing data online, you need to configure the path that contains the Go driver package.
- You can configure other Go environment variables based on your scenario parameters.

Run the **go env** command to view the Go environment variable configuration result and check whether the Go version is 1.13 or later.

- **Install the Go driver.**

- Obtain the Go driver package **GaussDB-Kernel_Database version number_OS information_64bit_Go.tar.gz** from the driver package and save it to the local PC. Decompress the package to obtain the Go driver source code package.
- Go to the root path of the Go driver code and run the **go mod tidy** command to download related dependencies. You need to configure **GOPATH=\${Path for storing the Go driver dependency package}** in the environment variables.
- If the dependencies have been downloaded to the localhost, you can add a line "Replace the Go driver package with the local Go driver package address through replace" to **go.mod**, indicating that all import Go driver packages in the code are stored in the local path and the dependencies are not downloaded from the proxy.

 **CAUTION**

- When you run the **go mod tidy** command to download dependencies, some of them may be downloaded as an earlier version. If the earlier version has vulnerabilities, you can change the dependency version in the **go.mod** file and update the dependency to the version after the vulnerability is fixed to avoid risks.
 - Users are not involved in the driver development and can call Go driver 1.13 or later. The runtime library needs to be updated to 1.18 or later.
-

Driver Class

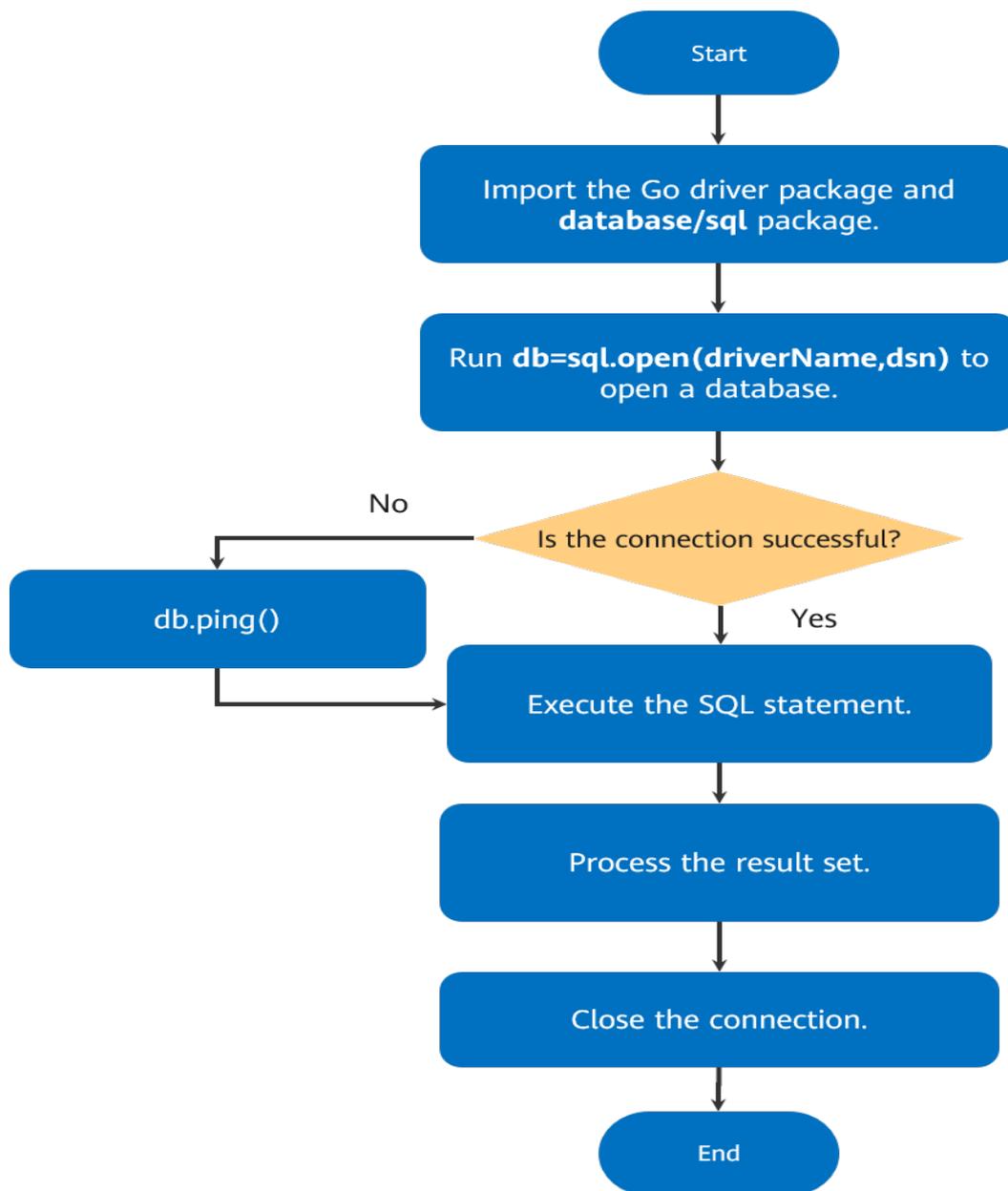
When creating a database connection, you need to enter the database driver name **gaussdb**.

NOTICE

The Go driver provided by the database does not adapt to mature ORM frameworks (such as XORM) in the industry.

5.9.2 Development Process

The Go driver of the database complies with the rule of the Go language third-party library. You only need to import the driver to the application program and save the driver code in the directory specified by **GOPATH**.

Figure 5-8 Application development process based on Go

According to [Figure 5-8](#), the Go driver application development process depends on the native database or SQL package of Go driver and the driver developed by GaussDB. The database or SQL package provides standard APIs that are used for the implementation of GaussDB for users.

5.9.3 Connecting to a Database

The Go driver provides the following method to generate a database connection object:

```
func Open(driverName, dataSourceName string) (*DB, error)
```

Parameter description:

- **driverName** indicates the driver name. The database driver name is **opengauss**, which is compatible with Postgres.
- **dataSourceName** indicates the data source to be connected. The value can be in DSN or URL format.
 - DSN format: key1 = value1 key2 = value2... Different groups of keywords are separated by space. The space on the left and right of the equal sign (=) is optional.
 - URL format: driverName://[userspec@][hostspec][/dbname][?paramspec]In the preceding information, **driverName** indicates the driver name. The database driver name is "**opengauss**", which is compatible with "**postgres**" and "**postgresql**".

userspec indicates user[:password]. When a URL is used for connection, the password cannot contain separators in the URL string. If the password contains separators, the DSN format is recommended.

hostspec indicates [host][:port][, ...].

dbname indicates the database name. Note: The initial user cannot be used for remote login.

paramspec indicates name=value[&...].

NOTICE

- In the DSN format, if there are multiple IP addresses:
 - When the value of **num(ip)** is the same as that of **num(port)**, the IP address matches the port number.
 - When the value of **num(ip)** is greater than that of **num(port)**, the IP address that cannot match the port number matches the first port number. For example, the matching condition of **host = ip1, ip2, ip3 port = port1, port2** is **ip1:port1, ip2:port2, ip3:port1**.
 - When the value of **num(ip)** is smaller than that of **num(port)**, the extra port numbers are discarded. For example, the mapping result of **host = ip1, ip2, ip3 port = port1, port2, port3, port4** is **ip1:port1, ip2:port2, ip3:port3**.
- In the URL format, if there are multiple IP addresses:
 - In the URL, *ip.port* must appear in pairs, that is, the value of **num(ip)** is the same as that of **num(port)**. Use commas (,) to separate multiple pairs. Example: **opengauss://user:password@ip1:port1, ip2:port2, ip3:port3/postgres**.
 - The URL contains only multiple IP addresses. The port number is specified by the environment variable or uses the default value **5432**. For example, in the case of **opengauss://user:password@ip1, ip2, ip3/postgres**, if the environment variable is set as **PGPORT = "port1, port2"**, the mapping is **ip1:port1, ip2:port2, ip3:port1**. If the environment variable is not set, the mapping is **ip1:5432, ip2:5432, ip3:5432**.

Parameters

Table 5-95 Database connection parameters

Parameter	Description
host	IP address of the host server, which can also be specified by the environment variable <code>PGHOST</code>
port	Port number of the host server, which can also be specified by the environment variable <code>PGPORT</code>
dbname	Database name, which can also be specified by the environment variable <code>PGDATABASE</code>
user	Username to be connected, which can also be specified by the environment variable <code>PGUSER</code>
password	Password of the user to be connected
connect_timeout	Timeout interval for connecting to the server, which can also be specified by the environment variable <code>PGCONNECT_TIMEOUT</code>
sslmode	SSL encryption mode, which can also be specified by the environment variable <code>PGSSLMODE</code> Value range: <ul style="list-style-type: none">● disable: SSL connection is disabled.● allow: If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.● prefer: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.● require: SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.● verify-ca: SSL connection is required, and whether the server certificate is issued by a trusted CA is verified.● verify-full: SSL connection is required, and whether the server certificate is issued by a trusted CA and whether the host name of the server is the same as that in the certificate are verified.
sslkey	Key location of the client certificate. If SSL connection is required and this parameter is not specified, you can set the environment variable <code>PGSSLKEY</code> to specify the location.

Parameter	Description
sslcert	File name of the client SSL certificate, which can also be specified by the environment variable $\$ \{PGSSLCERT\}$
sslrootcert	Name of the file that contains the SSL CA certificate, which can also be specified by the environment variable $\$ \{PGSSLROOTCERT\}$
sslcrl	File name of the SSL CRL. If a certificate listed in this file exists, the server certificate authentication will be rejected and the connection will fail. The value can also be specified by the environment variable $\$ \{PGSSLCRL\}$.
sslpassword	<p>Passphrase used to decrypt a key into plaintext. If this parameter is specified, the SSL key is an encrypted file. Currently, the SSL key supports DES encryption and AES encryption.</p> <p>NOTE The DES encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.</p>
disable_prepared_binary_result	<p>The value of this parameter is a string. If it is set to yes, the connection should not use the binary format when the query results are received from prepared statements. This parameter is used only for debugging.</p> <p>Value range: yes and no.</p>
binary_parameters	Specifies whether []byte is always sent in binary format. The value is a string. Value range: yes and no . If this parameter is set to yes , you are advised to bind parameters based on []byte to reduce internal type conversion.

Parameter	Description
target_session_attrs	<p>Connection type of the database, which can also be specified by the environment variable <code>PGTARGETSESSIONATTRS</code>. This parameter is used to identify the primary and standby nodes. There are six value options, namely, any, master, slave, preferSlave, read-write, and read-only. The default value is any.</p> <ul style="list-style-type: none"> ● any: attempts to connect to any DN in the URL connection string. ● master: attempts to connect to a primary node in the URL connection string. If the primary node cannot be found, an exception is thrown. ● slave: attempts to connect to a standby node in the URL connection string. If the standby node cannot be found, an exception is thrown. ● preferSlave: attempts to connect to a standby DN (if available) in the URL connection string. Otherwise, it connects to the primary DN. ● read-write: specifies that only the primary node can be connected. ● read-only: specifies that only the standby node can be connected.
loggerLevel	<p>Log level, which is used to print debugging information. The value can also be specified by the environment variable <code>PGLOGGERLEVEL</code>. The value can be trace, debug, info, warn, error, or none, in descending order of priority.</p>
application_name	<p>Name of the Go driver that is being connected. The default value is go-driver. You are advised not to configure this parameter.</p>
RuntimeParams	<p>Runtime parameter to be set to the session default value on a connection, for example, search_path, application_name, or timezone. For details about the parameters, see the default settings of the client connection. You can run the SHOW command to check whether the parameters are set successfully.</p>

Parameter	Description
autoBalance	<p>Character string type. Use this parameter to enable load balancing connections in the distributed environment. The value can be true, balance, roundrobin, shuffle, priorityn, or false. The default value is false.</p> <ol style="list-style-type: none"> 1. If this parameter is set to true, balance, or roundrobin, the Go SQL load balancing function is enabled to balance multiple connections of an application to each CN available in the cluster. Example: gauss://user:password@host1:port1,host2:port2/database?autoBalance=true The driver periodically obtains the list of available CNs in the entire cluster. For example, the obtained list is host1:port1,host2:port2,host3:port3,host4:port4. The refreshCNlpsTime parameter specifies the interval for obtaining the list, and the default value is 10s. When autoBalance is enabled on host1 and host2, HA is implemented only for the first connection. The driver will select available CNs from host1, host2, host3, and host4 in sequence to update the available CN list and Connector.Connect will be called on host1, host2, host3, and host4 using the RoundRobin algorithm. 2. priorityn indicates that the driver priority-based load balancing function is enabled. Multiple connections of an application are balanced to the first <i>n</i> available CNs configured in the URL. When the first <i>n</i> CNs are unavailable, connections are randomly allocated to other available CNs in the database cluster. <i>n</i> is a number not less than 0 and less than the number of CNs configured in the URL. Example: gauss://user:password@host1:port1,host2:port2,host3:port3/database?autoBalance=priority2 The driver periodically obtains the list of available CNs in the entire cluster (defined by refreshCNlpsTime). For example, the obtained list is host1:port1,host2:port2,host3:port3,host4:port4,host5:port5,host6:port6, where host1 and host2 are in AZ1, and host3 and host4 are in AZ2.

Parameter	Description
	<p>The driver preferentially selects host1 and host2 for load balancing. If both host1 and host2 are unavailable, the driver randomly selects a CN from host3, host4, host5, and host6 for connection.</p> <p>3. If this parameter is set to shuffle, the driver random load balancing is enabled to randomly and evenly distribute multiple connections of the application to available CNs in the database cluster. Example: gauss://user:password@host1:port1,host2:port2,host3:port3/database?autoBalance=shuffle</p> <p>The driver periodically obtains the list of available CNs in the entire cluster. For example, the obtained list is host1:port1,host2:port2,host3:port3,host4:port4. The refreshCNlpsTime parameter specifies the interval for obtaining the list, and the default value is 10s.</p> <p>For the first connection, host1:port1,host2:port2,host3:port3 is used for HA. For subsequent connections, the shuffle algorithm is used to randomly select a CN from the refreshed CN list.</p> <p>4. If this parameter is set to false in the centralized scenarios, the driver load balancing and priority-based load balancing functions are disabled. The default value is false.</p> <p>CAUTION Load balancing is based on the connection level rather than the transaction level. If the connection is persistent and the load on the connection is unbalanced, the load on the CN may be unbalanced. Load balancing can be used only in distributed scenarios and cannot be used in centralized scenarios.</p> <p>NOTE Load balancing is based on the connection level rather than the transaction level. If the connection is persistent and the load on the connection is unbalanced, the load on the CN may be unbalanced. Load balancing can be used only in distributed scenarios and cannot be used in centralized scenarios.</p>

Parameter	Description
recheckTime	Integer type. Use this parameter to specify the interval at which the driver periodically checks the status of CNs in the database cluster and obtains the IP address list of available CNs. The value ranges from 5s to 60s and the default value is 10s .
usingEip	Boolean type. This value specifies whether to use an elastic IP address for load balancing. The default value is true , indicating that an elastic IP address is used for load balancing. The value false indicates that a data IP address is used for load balancing.

Example 1:

```

package main
// Set the dependency package based on the dependency package path in the environment.
import (
    "context"
    "database/sql"
    "fmt"
    _ "gitee.com/opengauss/openGauss-connector-go-pq"
    "log"
    "strings"
    "time"
)
// Multiple IP addresses and ports (ip:port) are used as an example. In this example, the username and
password are stored in environment variables. Before running this example, set environment variables in the
local environment (set the environment variable name based on the actual situation).
func main() {
    ctx := context.Background()
    ctx2SecondTimeout, cancelFunc2SecondTimeout := context.WithTimeout(ctx, 2 * time.Second)
    defer cancelFunc2SecondTimeout()

    hostip1 := os.Getenv("GOHOSTIP1") // GOHOSTIP1 indicates the IP address written into the environment
variable.
    hostip2 := os.Getenv("GOHOSTIP2") // GOHOSTIP2 indicates the IP address written into the environment
variable.
    hostip3 := os.Getenv("GOHOSTIP3") // GOHOSTIP3 indicates the IP address written into the environment
variable.
    port1 := os.Getenv("GOPORT1") // GOPORT1 indicates the port number written into the environment
variable.
    port2 := os.Getenv("GOPORT2") // GOPORT2 indicates the port number written into the environment
variable.
    username := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the
environment variable.
    passwd := os.Getenv("GOPASSWD") // GOPASSWDW indicates the user password written into the
environment variable.

    str := "host="+hostip1+","+hostip2+","+hostip3+" port="+port1+","+port2+" user="+username+"
password="+passwd+" dbname=postgres sslmode=disable" // DSN connection string.
    //str := "opengauss://" + username + ":" + passwd + "@" + hostip1 + ":" + port1 + "," + hostip2 + ":" + port2 + "," + hostip3 + "/"
postgres?sslmode=disable" // URL connection string.
    db, err := sql.Open("opengauss", str)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    // Ping database connection with 2 second timeout

```

```
err = db.PingContext(ctx2SecondTimeout)
if err != nil {
    log.Fatal(err)
}

sqls := []string {
    "drop table if exists testExecContext",
    "create table testExecContext(f1 int, f2 varchar(20), f3 number, f4 timestamptz, f5 boolean)",
    "insert into testExecContext values(1, 'abcdefg', 123.3, '2022-02-08 10:30:43.31 +08', true)",
    "insert into testExecContext values(:f1, :f2, :f3, :f4, :f5)",
}

inF1 := []int{2, 3, 4, 5, 6}
inF2 := []string{"hello world", "Huawei", "Beijing 2022 Winter Olympics", "nanjing", "Research Center"}
inF3 := []float64{641.43, 431.54, 5423.52, 665537.63, 6503.1}
inF4 := []time.Time{
    time.Date(2022, 2, 8, 10, 35, 43, 623431, time.Local),
    time.Date(2022, 2, 10, 19, 11, 54, 353431, time.Local),
    time.Date(2022, 2, 12, 6, 11, 15, 636431, time.Local),
    time.Date(2022, 2, 14, 4, 51, 22, 747653, time.Local),
    time.Date(2022, 2, 16, 13, 45, 55, 674636, time.Local),
}
inF5 := []bool{false, true, false, true, true}

for _, s := range sqls {
    if strings.Contains(s, ":f") {
        for i, _ := range inF1 {
            _, err := db.ExecContext(ctx2SecondTimeout, s, inF1[i], inF2[i], inF3[i], inF4[i], inF5[i])
            if err != nil {
                log.Fatal(err)
            }
        }
    } else {
        _, err = db.ExecContext(ctx2SecondTimeout, s)
        if err != nil {
            log.Fatal(err)
        }
    }
}

var f1 int
var f2 string
var f3 float64
var f4 time.Time
var f5 bool
err = db.QueryRowContext(ctx2SecondTimeout, "select * from testExecContext").Scan(&f1, &f2, &f3, &f4, &f5)
if err != nil {
    log.Fatal(err)
} else {
    fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
}

row, err := db.QueryContext(ctx2SecondTimeout, "select * from testExecContext where f1 > :1", 1)
if err != nil {
    log.Fatal(err)
}
defer row.Close()

for row.Next() {
    err = row.Scan(&f1, &f2, &f3, &f4, &f5)
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
    }
}
}
```

5.9.4 Connecting to a Database (Using SSL)

The Go driver supports SSL connections to the database. After the SSL mode is enabled, if the Go driver connects to the database server in SSL mode, the Go driver uses the standard TLS 1.3 protocol by default, and the TLS version must be 1.2 or later. This section describes how applications configure the client in SSL mode through the Go driver. For details about how to configure the server, contact the administrator. To use the method described in this section, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

NOTE

In SSL-based certificate authentication mode, you do not need to specify the user password in the connection string.

Configuring the Client

Upload the **client.key**, **client.crt**, and **ca-cert.pem** files generated during server configuration to the client. For details about server configuration, contact the administrator.

Example 1:

```
package main
// Set the dependency package based on the dependency package path in the environment.
import (
    "database/sql"
    "fmt"
    _ "gitee.com/opengauss/openGauss-connector-go-pq"
    "log"
)
// Mutual authentication is used as an example. In this example, the username and password are stored in
// environment variables. Before running this example, set environment variables in the local environment (set
// the environment variable name based on the actual situation).
func main() {
    hostip := os.Getenv("GOHOSTIP") // GOHOSTIP indicates the IP address written into the environment
    // variable.
    port := os.Getenv("GOPORT") // GOPORT indicates the port number written into the environment
    // variable.
    username := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the
    // environment variable.
    passwd := os.Getenv("GOPASSWD") // GOPASSWD indicates the user password written into the
    // environment variable.
    sslpasswd := os.Getenv("GOSSLPASSWD") // GOSSLPASSWD indicates the passphrase written into the
    // environment variable.
    dsnStr := "host=" + hostip + " port=" + port + " user=" + username + " password=" + passwd + "
    sslcert=certs/client.crt sslkey=certs/client.key sslpassword=" + sslpasswd
    parameters := []string {
        "sslmode=require",
        "sslmode=verify-ca sslrootcert=certs/ca-cert.pem",
    }

    for _, param := range parameters {
        db, err := sql.Open("opengauss", dsnStr+param)
        if err != nil {
            log.Fatal(err)
        }
    }

    var f1 int
    err = db.QueryRow("select 1").Scan(&f1)
    if err != nil {
        log.Fatal(err)
    } else {
```

```
    fmt.Printf("RESULT: select 1: %d\n", f1)
  }

  db.Close()
}
```

Example 2:

```
package main
// Set the dependency package based on the dependency package path in the environment.
import (
    "database/sql"
    _ "github.com/opengauss/openGauss-connector-go-pq"
    "log"
    "strings"
)
// For example, verify sslpassword. In this example, the username and password are stored in environment
// variables. Before running this example, set environment variables in the local environment (set the
// environment variable name based on the actual situation).
func main() {
    hostip := os.Getenv("GOHOSTIP") // GOHOSTIP indicates the IP address written into the environment
    // variable.
    port := os.Getenv("GOPORT") // GOPORT indicates the port number written into the environment
    // variable.
    username := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the
    // environment variable.
    passwd := os.Getenv("GOPASSWD") // GOPASSWD indicates the user password written into the
    // environment variable.
    dsnStr := "host=" + hostip + " port=" + port + " user=" + username + " password=" + passwd + "
    dbname=postgres"

    sslpasswd := os.Getenv("GOSSLPASSWD") // GOSSLPASSWD indicates the passphrase written into the
    // environment variable.
    connStrs := []string {
        " sslmode=verify-ca sslcert=certs/client_rsa.crt sslkey=certs/client_rsa.key sslpassword=" + sslpasswd + "
    sslrootcert=certs/cacert_rsa.pem",
        " sslmode=verify-ca sslcert=certs/client_ecdsa.crt sslkey=certs/client_ecdsa.key sslpassword=" + sslpasswd + "
    sslrootcert=certs/cacert_ecdsa.pem",
    }
    for _, connStr := range connStrs {
        db, err := sql.Open("opengauss", dsnStr + connStr)
        if err != nil {
            log.Fatal(err)
        }
        var f1 int
        err = db.QueryRow("select 1").Scan(&f1)
        if err != nil {
            if !strings.HasPrefix(err.Error(), "connect failed.") {
                log.Fatal(err)
            }
        }
    }
    db.Close()
}
```

5.9.5 Go APIs

5.9.5.1 sql.Open

The following table describes sql.Open.

Method	Description	Return Value
--------	-------------	--------------

Open(driverName, dataSourceName string)	Opens a database based on a specified database driver and the dedicated data source of the driver.	*DB and error
---	--	---------------

For details about driverName and dataSourceName, see [Connecting to a Database](#).

5.9.5.2 type DB

The following table describes type DB.

Method	Description	Return Value
(db *DB)Begin()	Starts a transaction. The isolation level of the transaction is determined by the driver.	*Tx and error
(db *DB)BeginTx(ctx context.Context, opts *TxOptions)	Starts a transaction with a specified transaction isolation level. A specified context is used until the transaction is committed or rolled back. If the context is canceled, the SQL package rolls back the transaction.	*Tx and error
(db *DB)Close()	Closes the database and releases all the opened resources.	error
(db *DB)Exec(query string, args ...interface{})	Performs an operation that does not return rows of data.	Result and error
(db *DB)ExecContext(ctx context.Context, query string, args ...interface{})	Performs an operation that does not return rows of data in a specified context.	Result and error
(db *DB)Ping()	Checks whether the database connection is still valid and establishes a connection if necessary.	error
(db *DB)PingContext(ctx context.Context)	Checks whether the database connection is still valid in a specified context and establishes a connection if necessary.	error
(db *DB)Prepare(query string)	Creates a prepared statement for subsequent queries or executions.	*Stmt and error

(db *DB)PrepareContext(ctx context.Context, query string)	Creates a prepared statement for subsequent queries or executions in a specified context.	*Stmt and error
(db *DB)Query(query string, args ...interface{})	Executes a query and returns multiple rows of data.	*Rows and error
(db *DB)QueryContext(ctx context.Context, query string, args ...interface{})	Executes a query and returns multiple rows of data in a specified context.	*Rows and error
(db *DB)QueryRow(query string, args ...interface{})	Executes a query that returns only one row of data.	*Row
(db *DB)QueryRowContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns only one row of data in a specified context.	*Row

NOTICE

1. The Query(), QueryContext(), QueryRow(), and QueryRowContext() APIs are usually used in query statements, such as SELECT. The Exec() API is used for executing operation statements. If query APIs are used to execute non-query statements, the execution result may be unexpected. Therefore, you are advised not to use the query APIs to execute non-query statements, such as UPDATE and INSERT.
2. The result of executing a query statement using a query API needs to be obtained through the Next() API in **type Rows**. If the result is not obtained through the Next() API, unexpected errors may occur.

Parameters

Parameter	Description
ctx	Specified context.
query	Executed SQL statement.
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see Examples .

opts	Transaction isolation level and transaction access mode. The transaction isolation level (opts.Isolation) supports sql.LevelReadUncommitted, sql.LevelReadCommitted, sql.LevelRepeatableRead, and sql.LevelSerializable. The transaction access mode (opts.ReadOnly) can be true (read only) or false (read write).
------	--

Examples

```
// In this example, the username and password are stored in environment variables. Before running this
// example, set environment variables in the local environment (set the environment variable names based on
// the actual situation).
package main
/* The location of the Go driver dependency package is set based on the configured go.mod. */
import (
    "database/sql"
    _ "gitee.com/opengauss/openGauss-connector-go-pq"
    "log"
)

func main() {
    hostip := os.Getenv("GOHOSTIP") // GOHOSTIP indicates the IP address written into the environment
    // variable.
    port := os.Getenv("GOPORT") // GOPORT indicates the port number written into the environment
    // variable.
    username := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the
    // environment variable.
    passwd := os.Getenv("GOPASSWD") // GOPASSWD indicates the user password written into the
    // environment variable.
    str := "host=" + hostip + " port=" + port + " user=" + username + " password=" + passwd + "
    dbname=postgres sslmode=disable"
    db, err := sql.Open("opengauss", str)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    err = db.Ping()
    if err != nil {
        log.Fatal(err)
    }
    _, err = db.Exec("drop table if exists testuser.test")
    _, err = db.Exec("create table test(id int, name char(10))")

    // Binding by location
    _, err = db.Exec("insert into test(id, name) values(:1, :2)", 1, "Zhang San")
    if err != nil {
        log.Fatal(err)
    }

    // Binding by name
    _, err = db.Exec("insert into test(id, name) values(:id, :name)", sql.Named("id", 1), sql.Named("name",
    "Zhang San"))
    if err != nil {
        log.Fatal(err)
    }
}
```

5.9.5.3 type Stmt

The following table describes type Stmt.

Method	Description	Return Value
(s *Stmt)Close()	Closes a specified prepared statement.	error

<code>(s *Stmt)Exec(args ...interface{})</code>	Executes a prepared statement with specified parameters and returns a Result value. The Prepare-Bind-Execute (PBE) feature is supported. PBE is a method of sending and executing queries. The CN can receive PBE packets through complex query protocols to execute statements.	Result and error
---	---	------------------

	<p>NOTE</p> <ol style="list-style-type: none">1. The placeholder of a prepared statement can be a dollar sign (\$) or a question mark (?).2. The number of placeholders in a prepared statement is determined by the database. When the number of table columns exceeds the database limit or does not match the number of current table columns, the server returns an error.3. Batch PBE processing supports addition, deletion, and modification. During the batch operation, the maximum length of a U packet is limited to 1 GB minus 1 byte, that is, 0x3fffffff bytes. If the length exceeds the limit, "bind message length XXX too long. This can be caused by very large or incorrect length specifications on InputStream parameters" will be reported.4. When a record is inserted, the performance of PBE deteriorates greatly compared with that of a single-query statement (conn.simpleExec). Therefore, you are advised to use single-query statements, rather than the PBE statement.5. After the underlying error processing of the driver is reconstructed, the PBE performance decreases by less than 5%.	
--	---	--

(s *Stmt)ExecContext(ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns a Result value.	Result and error
(s *Stmt)Query(args ...interface{})	Executes a prepared statement with specified parameters and returns *Rows as the query result.	*Rows and error
(s *Stmt)QueryContext(ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns *Rows as the query result.	*Rows and error
(s *Stmt)QueryRow(args ...interface{})	Executes a prepared statement with specified parameters and returns *Row as the result.	*Row
(s *Stmt)QueryRowContext (ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns *Row as the result.	*Row

NOTICE

1. The Query(), QueryContext(), QueryRow(), and QueryRowContext() APIs are usually used in query statements, such as SELECT. The Exec() API is used for executing operation statements. If query APIs are used to execute non-query statements, the execution result may be unexpected. Therefore, you are advised not to use the query APIs to execute non-query statements, such as UPDATE and INSERT.
2. The result of executing a query statement using a query API needs to be obtained through the Next() API in **type Rows**. If the result is not obtained through the Next() API, unexpected errors may occur.

Parameters

Parameter	Description
ctx	Specified context.
query	Executed SQL statement.

args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see Examples in "type DB."
------	--

5.9.5.4 type Tx

The following table describes type Tx.

Method	Description	Return Value
(tx *Tx)Commit()	Commits a transaction.	error
(tx *Tx)Exec(query string, args ...interface{})	Performs an operation that does not return rows of data.	Result and error
(tx *Tx)ExecContext(ctx context.Context, query string, args ...interface{})	Performs an operation that does not return rows of data in a specified context.	Result and error
(tx *Tx)Prepare(query string)	Creates a prepared statement for subsequent queries or executions. The returned statement is executed within a transaction and cannot be used when the transaction is committed or rolled back.	*Stmt and error
(tx *Tx)PrepareContext(ctx context.Context, query string)	Creates a prepared statement for subsequent queries or executions. The returned statement is executed within a transaction and cannot be used when the transaction is committed or rolled back. The specified context will be used in the preparation phase, not in the transaction execution phase. The statement returned by this method will be executed in the transaction context.	*Stmt and error

(tx *Tx)Query(query string, args ...interface{})	Executes a query that returns rows of data.	*Rows and error
(tx *Tx)QueryContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns rows of data in a specified context.	*Rows and error
(tx *Tx)QueryRow(query string, args ...interface{})	Executes a query that returns only one row of data.	*Row
(tx *Tx)QueryRowContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns only one row of data in a specified context.	*Row
(tx *Tx) Rollback()	Rolls back a transaction.	error
(tx *Tx)Stmt(stmt *Stmt)	Returns a transaction-specific prepared statement for an existing statement. Example: str, err := db.Prepare("insert into t1 values(:1, :2)") tx, err := db.Begin() res, err := tx.Stmt(str).Exec(1, "aaa")	*Stmt
(tx *Tx)StmtContext(ctx context.Context, stmt *Stmt)	Returns a transaction-specific prepared statement for an existing statement in a specified context.	*Stmt

NOTICE

1. The Query(), QueryContext(), QueryRow(), and QueryRowContext() APIs are usually used in query statements, such as SELECT. The Exec() API is used for executing operation statements. If query APIs are used to execute non-query statements, the execution result may be unexpected. Therefore, you are advised not to use the query APIs to execute non-query statements, such as UPDATE and INSERT.
2. The result of executing a query statement using a query API needs to be obtained through the Next() API in **type Rows**. If the result is not obtained through the Next() API, unexpected errors may occur.

Parameters

Parameter	Description
-----------	-------------

ctx	Specified context.
query	Executed SQL statement.
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see Examples in "type DB."
stmt	Existing prepared statement, which is generally the prepared statement returned by the PREPARE statement

5.9.5.5 type Rows

The following table describes type Rows.

Method	Description	Return Value
(rs *Rows)Close()	Closes Rows to stop the iteration of the data set.	error
(rs *Rows)ColumnTypes()	Returns column information.	[]*ColumnType and error
(rs *Rows)Columns()	Returns the name of each column.	[]string and error
(rs *Rows)Err()	Returns any errors that occur during iteration.	error
(rs *Rows)Next()	Prepares the next data row to be read with the Scan method. If there is an additional result set, true is returned. Otherwise, false is returned.	boolean
(rs *Rows)Scan(dest ...interface{})	Copies the columns of the current iterated row of data to the value specified by dest .	error
(rs *Rows)NextResultSet()	Specifies whether there is an additional result set.	boolean

Parameters

Parameter	Description
dest	The column to be queried needs to be copied to the value specified by this parameter.

5.9.5.6 type Row

The following table describes type Row.

Method	Description	Return Value
(r *Row)Scan(dest ...interface{})	Copies the columns in the current row of data to the value specified by dest .	error
(r *Row)Err()	Returns errors that occur during execution.	error

Parameter Description

Parameter	Description
dest	The column to be queried needs to be copied to the value specified by this parameter.

5.9.5.7 type ColumnType

The following table describes type ColumnType.

Method	Description	Return Value
(ci *ColumnType)DatabaseTypeName()	Returns the name of the column-type database system. If an empty string is returned, driver-type names are not supported.	error
(ci *ColumnType)DecimalSize()	Returns the scale and precision of the decimal type. If the value of ok is false , the specified type is unavailable or not supported.	precision, scale int64, ok boolean
(ci *ColumnType)Length()	Returns the length of the data column type. If the value of ok is false , the specified type does not have a length.	length int64, ok boolean
(ci *ColumnType)ScanType()	Returns a Go type that can be used for scanning by using Rows.Scan.	reflect.Type
(ci *ColumnType)Name()	Returns the name of a data column.	string

5.9.5.8 type Result

The following table describes type Result.

Method	Description	Return Value
(res Result)RowsAffected()	Returns the number of rows affected by the INSERT, DELETE, UPDATE, SELECT, MOVE, FETCH, and COPY operations.	int64 and error

5.10 Appendix

5.10.1 JDBC

5.10.1.1 Data Type Mapping

The index relationships among data types, Java variable types, and JDBC types are as follows:

Compatibility Mode	GaussDB Data Type	Java Variable Type	JDBC Type Index
ORA/MYSQL	oid	java.lang.Long	java.sql.Types.BIGINT
ORA/MYSQL	numeric	java.math.BigDecimal	java.sql.Types.NUMERIC
ORA/MYSQL	tinyint	java.lang.Integer	java.sql.Types.TINYINT
ORA/MYSQL	smallint	java.lang.Integer	java.sql.Types.SMALLINT
ORA/MYSQL	bigint	java.lang.Long	java.sql.Types.BIGINT
ORA/MYSQL	float4	java.lang.Float	java.sql.Types.REAL
ORA/MYSQL	float8	java.lang.Double	java.sql.Types.DOUBLE
ORA/MYSQL	char	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	character	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	bpchar	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	character varying	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	varchar	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	text	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	name	java.lang.String	java.sql.Types.VARCHAR

Compatibility Mode	GaussDB Data Type	Java Variable Type	JDBC Type Index
ORA/MYSQL	bytea	byte[]	java.sql.Types.BINARY
ORA/MYSQL	blob	java.sql.Blob	java.sql.Types.BLOB
ORA/MYSQL	clob	java.sql.Clob	java.sql.Types.CLOB
ORA/MYSQL	boolean	java.lang.Boolean	java.sql.Types.BIT
MYSQL	date	java.sql.Date	java.sql.Types.DATE
ORA/MYSQL	time	java.sql.Time	java.sql.Types.TIME
ORA/MYSQL	timetz	java.sql.Time	java.sql.Types.TIME
ORA/MYSQL	timestamp	java.sql.Timestamp	java.sql.Types.TIMESTAMP
ORA/MYSQL	smalldatetime	java.sql.Timestamp	java.sql.Types.TIMESTAMP
ORA/MYSQL	timestampz	java.sql.Timestamp	java.sql.Types.TIMESTAMP
ORA/MYSQL	refcursor	java.sql.ResultSet	java.sql.Types.REF_CURSOR java.sql.Types.OTHER

5.10.1.2 Log Management

The GaussDB JDBC driver uses log records to help solve problems encountered when the GaussDB JDBC driver is used in applications. GaussDB JDBC supports the following log management methods:

- Use the SLF4J log framework for interconnecting with applications.
- Use the JdkLogger log framework for interconnecting with applications.

SLF4J and JdkLogger are mainstream frameworks for Java application log management in the industry. For details about how to use these frameworks, see the official documents (SLF4J: <http://www.slf4j.org/manual.html>; JdkLogger: <https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html>).

Method 1: Use the SLF4J log framework for interconnecting with applications.

When a connection is set up, **logger=Slf4JLogger** is configured in the URL. This mode supports log management and control. The SLF4J can implement powerful log management and control functions through related configurations in files. This method is recommended.

The SLF4J may be implemented by using Log4j or Log4j2. When Log4j is used to implement SLF4J, you need to add the **log4j-*.jar**, **slf4j-api-*.jar**, and **slf4j-log4j-*.jar** packages (*varies according to versions). In addition, specify the path (absolute or relative path) of the **log4j.properties** configuration file.

If the Log4j2 is used to implement the SLF4J, you need to add the following JAR packages: **log4j-api-*.jar**, **log4j-core-*.jar**, **log4j-slf4j18-impl-*.jar**, and **slf4j-api-*.alpha1.jar** (*varies according to versions). In addition, specify the path (absolute or relative path) of the **log4j2.xml** configuration file.

NOTICE

This method depends on the general SLF4J APIs, such as **org.slf4j.LoggerFactory.getLogger(String name)**, **org.slf4j.Logger.debug(String var1)**, **org.slf4j.Logger.info(String var1)**, **org.slf4j.Logger.warn(String warn)**, and **org.slf4j.Logger.warn(String warn)**. If these APIs are changed, logs cannot be recorded.

Example:

```
// Note: The path of the log4j.properties or log4j2.xml file must be specified during calling.

// Use the log4j.properties file to configure the attributes of the log recorder, such as the log level and
// output target. These properties are loaded into the application by calling the configure method of
// PropertyConfigurator and passing the path of the log4j.properties file.
//PropertyConfigurator.configure("log4j.properties");

// Create the Log4j2 configuration source, specify the path of the log4j2.xml configuration file, and
// initialize the log system.
//ConfigurationSource source = new ConfigurationSource(new FileInputStream("log4j2.xml"));
//Configurator.initialize(null, source);

public static Connection GetConnection(String username, String passwd){

    String sourceURL = "jdbc:opengauss://$ip:$port/database?logger=Slf4JLogger";
    Connection conn = null;

    try{
// Create a connection.
        conn = DriverManager.getConnection(sourceURL,username,passwd);
        System.out.println("Connection succeed!");
    }catch (Exception e){
        e.printStackTrace();
        return null;
    }
    return conn;
}
```

log4j.properties configuration file:

```
log4j.logger.com.huawei.opengauss.jdbc=ALL, log_gsjdbc

# Default file output configuration
log4j.appender.log_gsjdbc=org.apache.log4j.RollingFileAppender
log4j.appender.log_gsjdbc.Append=true
log4j.appender.log_gsjdbc.File=gsjdbc.log
log4j.appender.log_gsjdbc.Threshold=TRACE
log4j.appender.log_gsjdbc.MaxFileSize=10MB
log4j.appender.log_gsjdbc.MaxBackupIndex=5
log4j.appender.log_gsjdbc.layout=org.apache.log4j.PatternLayout
log4j.appender.log_gsjdbc.layout.ConversionPattern=%d %p %t %c - %m%n
log4j.appender.log_gsjdbc.File.Encoding=UTF-8
```

log4j2.xml configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
  <appenders>
    <Console name="Console" target="SYSTEM_OUT">
```

```
<PatternLayout pattern="%d %p %t %c - %m%n"/>
</Console>
<File name="FileTest" fileName="test.log">
  <PatternLayout pattern="%d %p %t %c - %m%n"/>
</File>
<!-- JDBC driver log file output configuration. Log rewinding is supported. When the log size exceeds
10 MB, a new file is created. The new file is named in the format of yyyy-mm-dd-file ID. -->
<RollingFile name="RollingFileJdbc" fileName="gsjdbc.log" filePattern="%d{yyyy-MM-dd}-%i.log">
  <PatternLayout pattern="%d %p %t %c - %m%n"/>
  <Policies>
    <SizeBasedTriggeringPolicy size="10 MB"/>
  </Policies>
</RollingFile>
</appenders>
<loggers>
  <root level="all">
    <appender-ref ref="Console"/>
    <appender-ref ref="FileTest"/>
  </root>
  <!-- JDBC driver logs. The log level is all. All logs can be viewed and exported to the gsjdbc.log file. -->
  <logger name="com.huawei.opengauss.jdbc" level="all" additivity="false">
    <appender-ref ref="RollingFileJdbc"/>
  </logger>
</loggers>
</configuration>
```

NOTICE

- In the configuration file, you can specify the configuration paths for JDBC and upper-layer services to store JDBC logs and service logs to different files.
- When multiple GaussDB data sources are configured in the same project, if **logger** is set to **Slf4JLogger** in some services, it will affect the logging configuration of each other. In this case, you are advised to configure the connection strings in a unified manner.

Method 2: Use the JdkLogger log framework for interconnecting with applications.

You can use the **logging.properties** configuration file or directly configure the parameters in the sample code.

- The default Java logging framework stores its configurations in a file named **logging.properties**. Java installs the global configuration file in the folder in the Java installation directory. The **logging.properties** file can also be created and stored with a single project.

logging.properties configuration file:

```
# Specify the processing program as a file.
handlers= java.util.logging.FileHandler

# Specify the default global log level.
.level= ALL

# Specify the log output control standard.
java.util.logging.FileHandler.level=ALL
java.util.logging.FileHandler.pattern=gsjdbc.log
java.util.logging.FileHandler.limit=500000
java.util.logging.FileHandler.count=30
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
java.util.logging.FileHandler.append=false
```

- Configure the parameters in the sample code:

```
System.setProperty("java.util.logging.FileHandler.pattern","jdbc.log");
FileHandler fileHandler = new FileHandler(System.getProperty("java.util.logging.FileHandler.pattern"));
Formatter formatter = new SimpleFormatter();
fileHandler.setFormatter(formatter);
Logger logger = Logger.getLogger("com.huawei.opengauss.jdbc");
logger.addHandler(fileHandler);
logger.setLevel(Level.ALL);
logger.setUseParentHandlers(false);
```

Link Trace Function

The GaussDB JDBC driver provides the application-to-database link trace function to associate discrete SQL statements on the database side with application requests. This function requires application developers to implement the `com.huawei.opengauss.jdbc.log.Tracer` API class and specify the full name of the API implementation class in the URL.

URL example:

```
String url = "jdbc:opengauss://$ip:$port/database?traceInterfaceClass=xxx.xxx.xxx.OpenGaussTraceImpl";
```

The `com.huawei.opengauss.jdbc.log.Tracer` API class is defined as follows:

```
public interface Tracer {
    // Retrieves the value of traceId.
    String getTraceId();
}
```

The following is an example of the `com.huawei.opengauss.jdbc.log.Tracer` API implementation class:

```
import com.huawei.opengauss.jdbc.log.Tracer;

public class OpenGaussTraceImpl implements Tracer {
    private static MDC mdc = new MDC();

    private final String TRACE_ID_KEY = "traceId";

    public void set(String traceId) {
        mdc.put(TRACE_ID_KEY, traceId);
    }

    public void reset() {
        mdc.clear();
    }

    @Override
    public String getTraceId() {
        return mdc.get(TRACE_ID_KEY);
    }
}
```

The following is an example of context mapping which is used to store trace IDs generated for different requests:

```
import java.util.HashMap;

public class MDC {
    static final private ThreadLocal<HashMap<String, String>> threadLocal = new ThreadLocal<>();

    public void put(String key, String val) {
        if (key == null || val == null) {
            throw new IllegalArgumentException("key or val cannot be null");
        } else {
            if (threadLocal.get() == null) {
                threadLocal.set(new HashMap<>());
            }
        }
    }
}
```

```
        threadLocal.get().put(key, val);
    }
}

public String get(String key) {
    if (key == null) {
        throw new IllegalArgumentException("key cannot be null");
    } else if (threadLocal.get() == null) {
        return null;
    } else {
        return threadLocal.get().get(key);
    }
}

public void clear() {
    if (threadLocal.get() == null) {
        return;
    } else {
        threadLocal.get().clear();
    }
}
}
```

Take the service **traceld** as an example. The prerequisite is that the table **test_trace_id (id int,name varchar(20))** is created.

```
String traceld = UUID.randomUUID().toString().replaceAll("-", "");
OpenGaussTraceImpl openGaussTrace = new OpenGaussTraceImpl();
openGaussTrace.set(traceld);
Connection con = DriverManager.getConnection(url, user, password);
pstmt = con.prepareStatement("SELECT * FROM test_trace_id WHERE id = ?");
pstmt.setInt(1, 1);
pstmt.execute();
pstmt = con.prepareStatement("INSERT INTO test_trace_id VALUES(?,?)");
pstmt.setInt(1, 2);
pstmt.setString(2, "test");
pstmt.execute();
openGaussTrace.reset();
```

NOTE

- When the link trace function is used, the link function at the application layer is guaranteed by services.
- The application must expose the API for obtaining **traceld** to the JDBC and configure the API implementation class to the JDBC connection string.
- SQL statements of the same request must use the same **traceld**.
- The value of **traceld** transferred by the application cannot exceed 32 bytes. Otherwise, the extra bytes will be truncated.

5.10.1.3 FAQ

5.10.1.3.1 Incorrect batchMode Settings

Symptom

Set the URL parameters **batchMode** to **on** and **reWriteBatchedInserts** to **true**, and use JDBC to insert data in batches. As a result, an exception is thrown, indicating that the number of bound parameters is inconsistent with the number of parameters required by the statement.

bind message supplies * parameters, but prepared statement "" requires *

Example:

```
// conn is a created connection object. The URL parameters for creating the connection contain
&batchMode=on&rewriteBatchedInserts=true.
// Bind parameters in batches and then execute the statement. The number of bound parameters does not
// match the number of columns in the rewritten INSERT statement. As a result, an exception is reported.
// java.sql.BatchUpdateException: bind message supplies 3 parameters, but prepared statement "" requires 6

PreparedStatement stmt = conn.prepareStatement("INSERT INTO test_tbl VALUES (?, ?, ?)");

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbb");
stmt.addBatch();

stmt.executeBatch();
```

Cause Analysis

When **rewriteBatchedInserts** is set to **true**, the batch statement combines multiple SQL statements into one. As a result, the number of reserved parameter columns in the statement changes. If **batchMode** is set to **on**, parameters are bound based on the SQL statements before combination. As a result, the number of bound parameters is inconsistent with the number of parameters required by the statement.

Solution

If **rewriteBatchedInserts** is set to **true**, set **batchMode** to **off**.

5.10.1.3.2 Error Is Reported When Verification Is Enabled for Data Insertion in the Hibernate Framework

Symptom

The customer migrates data from the ORA-compatible database to GaussDB. The table structure is migrated using the DRS tool. After the migration, the original service code of the customer is unavailable, and an error is reported when the Hibernate framework verifies the table structure.

```
Schema-validation: wrong column type encountered in column [execute_time] in table [abnormal_event];
found [int8 (Types#BIGINT)], but expecting [int4 (Types#INTEGER)]
```

Example:

- The structure of the **Student** table in the ORA-compatible database is as follows:
id number(15,0)
sid number(15,0)
age number(10,0)
- After data is migrated to GaussDB using DRS, the **Student** table structure is as follows:

```
id bigint
sid bigint
age Integer
```

- When the **Student** entity class information in the actual database is as follows:

```
Long id
Long sid
Long age -- An error is reported when the Hibernate framework verifies the table structure.
```

Configuration file **hibernate.cfg.xml**:

```
<hibernate-configuration>
  <session-factory>
    <!-- GaussDB connection information -->
    <property name="connection.driver_class">com.huawei.opengauss.jdbc.Driver</property>
    <property name="connection.url">jdbc:opengauss://x.x.x.xx/postgres?currentSchema=public</
property>
    <property name="connection.username">xxxxxx</property>
    <property name="connection.password">xxxxxx</property>

    <!-- The following configurations are optional: -->

    <!-- Specify whether to support dialects. -->
    <!-- In pgSQL-compatible mode, the following configuration must exist: -->
    <property name="dialect">org.hibernate.dialect.PostgreSQL92Dialect</property>
    <!-- In ORA-compatible mode, you are advised to change the configuration as follows: -->
    <!-- <property name="dialect">org.hibernate.dialect.OracleDialect</property>-->

    <property name="met"></property>

    <!-- Specify whether to print SQL statements when CURD is executed. -->
    <property name="show_sql">>true</property>
    <!-- The table is created automatically. -->
    <!-- <property name="hbm2ddl.auto">create</property>-->
    <!-- Enable the verification when inserting data. -->
    <property name="hbm2ddl.auto">validate</property>
    <!-- Disable the verification. -->
    <!-- <property name="hbm2ddl.auto">none</property>-->

    <!-- Resource registration (entity class mapping file) -->
    <mapping resource="student.xml"/>
  </session-factory>
</hibernate-configuration>
```

Main function for inserting data:

```
// Create the object to be tested.
Student student = new Student();
student.setId(20L);
student.setAge(222L);
student.setSid(222L);

// Start a transaction and obtain data based on the session.
Configuration conf = new Configuration().configure();
SessionFactory sessionFactory = conf.buildSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();

// Save data using sessions.
session.save(student);

// Commit a transaction.
transaction.commit();

// After the operation is complete, close the session connection object.
session.close();
```

Cause Analysis

- If the data can be successfully inserted into the ORA-compatible database and passes the verification: When verifying data, the Hibernate framework compares **TypeCode** of the inserted data type `java.lang.Long` with that of the number type in the table structure. If they are inconsistent, the Hibernate framework compares **sqlType** of the inserted data of the long type with **Type** of the table structure and the long type is converted to `number(19,0)` (the mapping is maintained by the `org.hibernate.dialect.OracleDialect`). The type of the table structure is number, and `number(19,0)` starts with **number**. Therefore, the verification is successful. When the ORA-compatible database is used, the service code involves data of the integer type. You can use the `java.lang.Long` type in Java to insert the data.
- If the GaussDB verification is different and the data is inserted successfully after the verification is disabled: When the Hibernate framework verifies the inserted data, the service code is of the `java.lang.Long` type, which corresponds to **TypeCode** of the table data type `bigint`. Therefore, no error is reported. If the table data type is integer, the system checks **TypeCode** of the `java.lang.Long` type in the service code and compares it with that of the integer type in the table data. If they are inconsistent, the system converts the long type to `int8` based on the `org.hibernate.dialect.PostgreSQLDialect` mapping, the table data type integer is converted to `int4`, which cannot be matched. As a result, the verification fails. When GaussDB is used, the service code involves data of the integer type. Therefore, the Java integer corresponding to the table type must be used.

Solution

You can use the following methods to rectify the fault:

- Disable the verification function.
- Modify the service code at the customer side and use the corresponding Java integer for different table data types.

5.10.1.3.3 Error Is Reported or Connection Is Blocked in SSL Mode

Symptom

When JDBC establishes a connection in SSL mode, a strong random number is obtained on the client. During the connection establishment, the error information may be displayed in different scenarios.

Scenario 1: Error report.

```
"Thread-0" #18 prio=5 os_prio=0 tid=0x00007f2ad0385000 nid=0x5429 runnable [0x00007f2aa069b000]
java.lang.Thread.State: RUNNABLE
    at java.io.FileInputStream.readBytes(Native Method)
    at java.io.FileInputStream.read(FileInputStream.java:255)
    at sun.security.provider.NativePRNG$RandomIO.readFully(NativePRNG.java:424)
    at sun.security.provider.NativePRNG$RandomIO.ensureBufferValid(NativePRNG.java:526)
    at sun.security.provider.NativePRNG$RandomIO.implNextBytes(NativePRNG.java:545)
    - locked <0x000000067273a950> (a java.lang.Object)
    at sun.security.provider.NativePRNG$RandomIO.access$400(NativePRNG.java:331)
    at sun.security.provider.NativePRNG$Blocking.engineNextBytes(NativePRNG.java:268)
    at java.security.SecureRandom.nextBytes(SecureRandom.java:468)
    at java.security.SecureRandom.next(SecureRandom.java:491)
    at java.util.Random.nextInt(Random.java:329)
    at sun.security.ssl.SSLContextImpl.engineInit(SSLContextImpl.java:106)
    at javax.net.ssl.SSLContext.init(SSLContext.java:282)
    at org.postgresql.ssl.LibPQFactory.<init>(LibPQFactory.java:175)
    at org.postgresql.core.SocketFactoryFactory.getSslSocketFactory(SocketFactoryFactory.java:62)
    at org.postgresql.ssl.MakeSSL.convert(MakeSSL.java:33)
    at org.postgresql.core.v3.ConnectionFactoryImpl.enableSSL(ConnectionFactoryImpl.java:723)
    at org.postgresql.core.v3.ConnectionFactoryImpl.tryConnect(ConnectionFactoryImpl.java:203)
    at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:330)
    at org.postgresql.core.ConnectionFactory.openConnection(ConnectionFactory.java:58)
    at org.postgresql.jdbc.PgConnection.<init>(PgConnection.java:357)
```

Scenario 2: The connection is blocked. If **loginTimeout** is set in the connection string, the message "Connection attempt timed out" is displayed. If this parameter is not set, the connection is blocked.

Cause Analysis

The random number generation on the client is too slow to meet product requirements. The entropy source is insufficient. As a result, the service fails to be started. This problem exists in some Linux environments.

Solution

- Method 1: Start the haveged service on the client and increase the entropy value of the system entropy pool to improve the speed of reading random numbers.

The startup command is as follows:
systemctl start haveged

- Method 2: Adjust the JDK configuration on the client.

Open the `$(AVA_PATH)/jre/lib/security/java.security` file and modify the following configuration items:

```
securerandom.source=file:/dev/.urandom
securerandom.strongAlgorithms=NativePRNGNonBlocking:SUN
```

NOTE

The essence of method 2 is to use pseudo-random numbers instead of strong random numbers to reduce the entropy value to be consumed. All applications that use the JDK on the client are affected. Pseudo-random numbers are used to instead of strong random numbers.

5.10.2 libpq

5.10.2.1 Connection Parameters

Table 5-96 Connection parameters

Parameter	Description
host	<p>Name of the host to connect to. If the host name starts with a slash (/), UDS communications instead of TCP/IP communications are used. The value is the directory where the socket file is stored. If host is not specified, the default behavior is to connect to the UDS in the /tmp directory (or the socket directory specified during GaussDB installation). On a machine without a UDS, the default behavior is to connect to localhost.</p> <p>You can specify multiple host names by using a character string separated by commas (.). Multiple host names can be specified.</p>
hostaddr	<p>IP address of the host to connect to. The value is in standard IPv4 address format, for example, 172.28.40.9. If a non-null character string is specified, TCP/IP communications are used.</p> <p>You can specify multiple IP addresses by using a character string separated by commas (.). Multiple IP addresses can be specified.</p> <p>Replacing host with hostaddr can prevent applications from querying host names, which may be important for applications with time constraints. However, a host name is required for GSSAPI or SSPI authentication methods. Therefore, the following rules are used:</p> <ol style="list-style-type: none"> 1. If host is specified but hostaddr is not, a query for the host name will be executed. 2. If hostaddr is specified but host is not, the value of hostaddr is the server network address. If the host name is required for authentication, the connection attempt fails. 3. If both host and hostaddr are specified, the value of hostaddr is the server network address. The value of host is ignored unless it is required by authentication, in which case it is used as the host name. <p>NOTICE</p> <ul style="list-style-type: none"> • If host is not the server name in the network address specified by hostaddr, the authentication may fail. • If neither host nor hostaddr is specified, libpq will use a local UDS for connection. If the machine does not have a UDS, it will attempt to connect to localhost.
port	<p>Port number of the host server, or the socket file name extension for UDS connections.</p> <p>You can specify multiple port numbers by using a character string separated by commas (.). Multiple port numbers can be specified.</p>
user	<p>Name of the user to be connected. By default, the username is the same as the OS name of the user running the application.</p>
dbname	<p>Database name. The default value is the same as the username.</p>

Parameter	Description
password	Password to be used if the server requires password authentication.
connect_timeout	Maximum timeout period of the connection, in seconds (written as a decimal integer string). The value 0 or null indicates infinity. You are advised not to set the connection timeout period to a value less than 2 seconds.
client_encoding	Client encoding for the connection. In addition to the values accepted by the corresponding server options, you can use auto to determine the correct encoding from the current environment in the client (the <i>LC_CTYPE</i> environment variable in the Unix system).
tty	This parameter can be ignored. (This parameter was used to specify the location to which the debugging output of the server was sent).
options	Adds command-line options to send to the server at runtime.
application_name	Specifies the current user identity for the application_name parameter.
fallback_application_name	Specifies a backup value for the application_name parameter. This value is used if no value is set for application_name through a connection parameter or the <i>PGAPPNAME</i> environment variable. In a common tool program, if you set a default name but do not want the default name to be overwritten by the user, you can specify a backup value.
keepalives	Specifies whether TCP keepalive is enabled on the client side. The default value is 1 , indicating that the function is enabled. The value 0 indicates that the function is disabled. Ignore this parameter for UDS connections.
keepalives_idle	The number of seconds of inactivity after which TCP should send a keepalive message to the server. The value 0 indicates that the default value is used. Ignore this parameter for UDS connections or if keepalive is disabled.
keepalives_interval	The number of seconds after which a TCP keepalive message that is not acknowledged by the server should be retransmitted. The value 0 indicates that the default value is used. Ignore this parameter for UDS connections or if keepalive is disabled.
keepalives_count	Controls the number of times that keepalive messages are sent through TCP. The value 0 indicates that the default value is used. Ignore this parameter for UDS connections or if keepalive is disabled.

Parameter	Description
tcp_user_timeout	Specifies the maximum duration for which transmitted data can remain unacknowledged before the TCP connection is forcibly closed on an OS that supports the TCP_USER_TIMEOUT socket option. The value 0 indicates that the default value is used. Ignore this parameter for UDS connections.
rw_timeout	Sets the read and write timeout interval of the client connection.
sslmode	Specifies whether to enable SSL encryption. <ul style="list-style-type: none"> • disable: SSL connection is disabled. • allow: If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified. • prefer: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified. • require: SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified. • verify-ca: SSL connection is required. Currently, Windows ODBC does not support certificate-based authentication. • verify-full: SSL connection is required. Currently, Windows ODBC does not support certificate-based authentication.
sslcompression	If this parameter is set to 1 (default value), the data transmitted over the SSL connection is compressed (this requires that the OpenSSL version be 0.9.8 or later). If it is set to 0 , compression will be disabled (this requires OpenSSL 1.0.0 or later). If a connection without SSL is established, this parameter is ignored. If the OpenSSL version in use does not support this parameter, it will also be ignored. Compression takes up CPU time, but it increases throughput when the bottleneck is the network. If CPU performance is a limiting factor, disabling compression can improve response time and throughput.
sslcert	This parameter specifies the file name of the client SSL certificate. If no SSL connection is established, this parameter is ignored.
sslkey	This parameter specifies the location of the key used for the client certificate. It can specify a key obtained from an external "engine" (the engine is a loadable module of OpenSSL). The description of an external engine should consist of a colon-separated engine name and an engine-related key identifier. If no SSL connection is established, this parameter is ignored.
sslrootcert	This parameter specifies the name of a file that contains the SSL Certificate Authority (CA) certificate. If the file exists, the system authenticates the server certificate issued by one of these authorities.

Parameter	Description
sslcr	This parameter specifies the file name of the SSL Certificate Revocation List (CRL). If a certificate listed in this file exists, the server certificate authentication will be rejected.
requirepeer	This parameter specifies the OS user of the server, for example, requirepeer=postgres . When a UDS connection is established, if this parameter is set, the client checks whether the server process is running under the specified username at the beginning of the connection. If not, the connection will be interrupted by an error. This parameter can be used to provide server authentication similar to that of the SSL certificate on TCP/IP connections. (Note that if the UDS is in /tmp or another public writable location, any user can start a server for listening to the location. Use this parameter to ensure that you are connected to a server that is run by a trusted user.) This option is supported only on platforms that implement the peer authentication method.
krbsrvname	This parameter specifies the Kerberos service name used for GSSAPI authentication. For successful Kerberos authentication, this value must match the service name specified in the server configuration.
gsslib	This parameter specifies the GSS library used for GSSAPI authentication. It is used only in the Windows OS. If this parameter is set to gssapi , the libpq is forced to use the GSSAPI library to replace the default SSPI for authentication.
service	This parameter specifies the name of the service for which the additional parameter is used. It specifies a service name in pg_service.conf that holds the additional connection parameters. This allows the application to specify only one service name so that the connection parameters can be centrally maintained.
authtype	authtype is no longer used, so it is marked as a parameter not to be displayed. It is retained in an array so as not to reject the conninfo string from old applications that might still try to set it.
remote_node_name	Specifies the name of the remote node connected to the local node.
localhost	Specifies the localhost in a connection channel.
localport	Specifies the local port in a connection channel.
fencedUdfRPCMode	Specifies whether the fenced UDF RPC protocol uses UDSs or special socket file names. The default value is 0 , indicating that the UDS mode is used and the file type is .s.PGSQL.%d . To use the fenced UDF mode, set this parameter to 1 . In this case, the file type is .s.fencedMaster_unixdomain .

Parameter	Description
replication	<p>Specifies whether the connection should use replication protocols instead of common protocols. Protocols with this parameter configured are internal protocols used for PostgreSQL replication connections and tools such as pg_basebackup, while they can also be used by third-party applications. The following values, which are case-insensitive, are supported:</p> <ul style="list-style-type: none"> • true, on, yes, and 1 Specify whether the physical replication mode is connected. • database Specifies that the logical replication mode and the database specified by dbname are connected. • false, off, no, and 0 Specify that the connection is a regular connection, which is the default behavior. <p>In physical or logical replication mode, only simple query protocols can be used.</p>
backend_version	Specifies the backend version to be passed to the remote end.
prototype	Sets the current protocol level. The default value is PROTO_TCP .
enable_ce	Specifies whether a client is allowed to connect to a fully-encrypted database. The default value is 0 (disabled). To enable the encrypted equality query, set this parameter to 1 .
connection_info	<p>The value of connection_info is a JSON character string consisting of driver_name, driver_version, driver_path, and os_user.</p> <p>If the value is not null, use connection_info and ignore connectionExtraInf.</p> <p>If the value is null, a connection information string related to libpq is generated. When connectionExtraInf is set to false, the value of connection_info consists of only driver_name and driver_version.</p>
connectionExtraInf	Specifies whether the value of connection_info contains extension information. The default value is 0 . If the value contains other information, set this parameter to 1 .

Parameter	Description
target_session_attrs	<p>Specifies the type of the host to be connected. The connection is successful only when the host type is the same as the configured value. This parameter is verified only when multiple IP addresses are specified. The rules for setting target_session_attrs are as follows:</p> <ul style="list-style-type: none"> • any: All types of hosts can be connected. • read-write: The connection is set up only when the connected host is readable and writable. • read-only: Only readable hosts can be connected. • primary (default value): Only the primary node in the primary/standby system can be connected. • standby: Only the standby node in the primary/standby system can be connected. • prefer-standby: The system first attempts to find a standby node for connection. If all hosts in the hosts list fail to be connected, try the any mode.

5.10.3 Parameters Related to Log Output

To control the output of log files and better understand the operating status of the database, modify specific configuration parameters in the **postgresql.conf** file in the instance data directory.

Table 5-97 describes the adjustable configuration parameters.

Table 5-97 Configuration parameters

Parameter	Description	Value Range	Remarks
client_min_messages	Level of messages to be sent to clients.	<ul style="list-style-type: none"> • DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • LOG • NOTICE • WARNING • ERROR • FATAL • PANIC Default value: NOTICE	Messages of the set level or lower will be sent to clients. The lower the level is, the fewer the messages will be sent.

Parameter	Description	Value Range	Remarks
log_min_messages	Level of messages to be recorded in server logs.	<ul style="list-style-type: none"> • DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • LOG • FATAL • PANIC Default value: WARNING	Messages higher than the set level will be recorded in logs. The higher the level is, the fewer the server logs will be recorded.
log_min_error_statement	Level of SQL error statements to be recorded in server logs.	<ul style="list-style-type: none"> • DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • FATAL • PANIC Default value: ERROR	SQL error statements of the set level or higher will be recorded in server logs. Only a system administrator is allowed to modify this parameter.

Parameter	Description	Value Range	Remarks
log_min_duration_statement	Minimum execution duration of a statement. If the execution duration of a statement is equal to or longer than the set milliseconds, the statement and its duration will be recorded in logs. Enabling this function can help you track the query attempts to be optimized.	INT type Default value: -1 Unit: millisecond	The value -1 indicates that the function is disabled. Only a system administrator is allowed to modify this parameter.
log_connections/ log_disconnections	Specifies whether to record a server log message when each session is connected or disconnected.	<ul style="list-style-type: none"> ● on: The system records a log server when each session is connected or disconnected. ● off: The system does not record a log server when each session is connected or disconnected. Default value: off	-
log_duration	Specifies whether to record the duration of each executed statement.	<ul style="list-style-type: none"> ● on: The system records the duration of each executed statement. ● off: The system does not record the duration of each executed statement. Default value: off	Only a system administrator is allowed to modify this parameter.

Parameter	Description	Value Range	Remarks
log_statement	SQL statements to be recorded in logs.	<ul style="list-style-type: none"> • none: The system does not record any SQL statements. • ddl: The system records data definition statements. • mod: The system records data definition statements and data operation statements. • all: The system records all statements. Default value: none	Only a system administrator is allowed to modify this parameter.
log_hostname	Specifies whether to record host names.	<ul style="list-style-type: none"> • on: The system records host names. • off: The system does not record host names. Default value: off	By default, connection logs only record the IP addresses of connected hosts. With this function, the host names will also be recorded. This parameter has an impact on viewing audit results, PG_STAT_ACTIVITY , and the GUC parameter log_line_prefix .

Table 5-98 describes the preceding parameter levels.

Table 5-98 Description of log level parameters

Level	Description
DEBUG[1-5]	Provides information that can be used by developers. Level 1 is the lowest level whereas level 5 is the highest level.
INFO	Provides information about users' hidden requests, for example, information about the VACUUM VERBOSE process.

Level	Description
NOTICE	Provides information that may be important to users, for example, truncations of long identifiers or indexes created as a part of a primary key.
WARNING	Provides warning information for users, for example, COMMIT out of transaction blocks.
ERROR	Reports an error that causes a command to terminate.
LOG	Reports information that administrators may be interested in, for example, the activity levels of check points.
FATAL	Reports the reason that causes a session to terminate.
PANIC	Reports the reason that causes all sessions to terminate.

6 SQL Optimization

The aim of SQL optimization is to maximize the utilization of resources, including CPU, memory, disk I/O, and network I/O. All optimization methods are intended for resource utilization. To maximize resource utilization is to run SQL statements as efficiently as possible to achieve the highest performance at a lower cost. For example, when performing a typical point query, you can use a combination of Seq Scan and filter (that is, read each tuple and match the point query condition). You can also use Index Scan, which can implement the query at a lower cost but achieve the same effect.

You can determine a proper cluster deployment solution and table definition based on hardware resources and service characteristics. This is the basis of meeting performance requirements. The following performance tuning sections assume that you have finished installation based on a proper cluster solution in the software installation guide and performed database design based on the guide for database design and development.

6.1 Query Execution Process

The process from receiving SQL statements to the statement execution by the SQL engine is shown in [Figure 6-1](#) and described in [Table 6-1](#). The texts in red are steps where database administrators can optimize queries.

Figure 6-1 Execution process of query-related SQL statements by the SQL engine

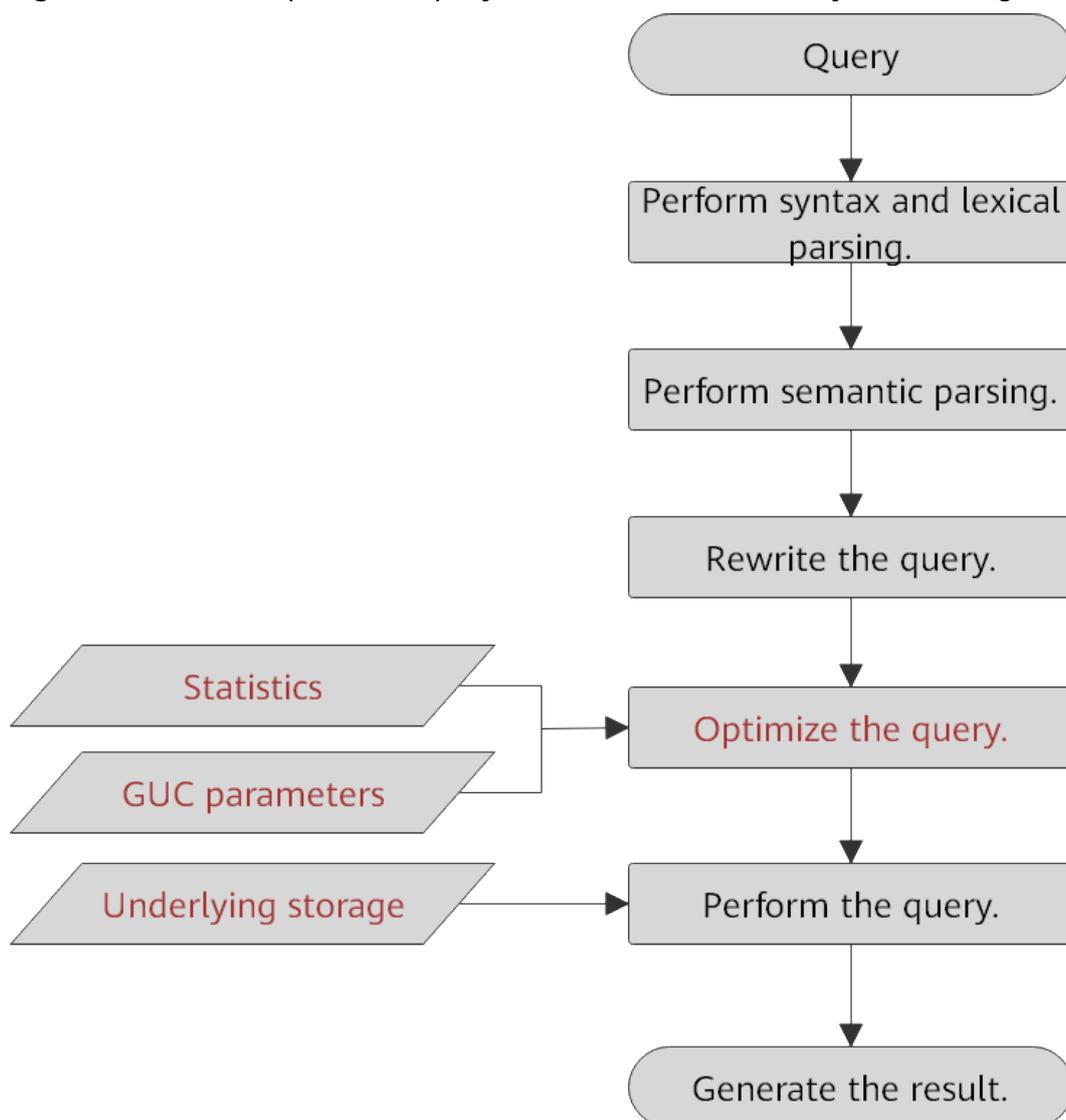


Table 6-1 Execution process of query-related SQL statements by the SQL engine

Step	Description
1. Perform syntax and lexical parsing.	Converts the input SQL statements from the string data type to the formatted structure stmt based on the specified SQL statement rules.
2. Perform semantic parsing.	Converts the formatted structure obtained from the previous step into objects that can be recognized by the database.
3. Rewrite the query statements.	Converts the output of the previous step into the structure that optimizes the query execution.

Step	Description
4. Optimize the query.	Determines the execution mode of SQL statements (the execution plan) based on the result obtained from the previous step and the internal database statistics. For details about how the internal database statistics and GUC parameters affect the query optimization (execution plan), see Optimizing Queries Using Statistics and Optimizing Queries Using GUC Parameters .
5. Perform the query.	Executes the SQL statements based on the execution path specified in the previous step. Selecting a proper underlying storage mode improves the query execution efficiency.

Optimizing Queries Using Statistics

The GaussDB optimizer is a typical cost-based optimization (CBO). By using CBO, the database calculates the number of tuples and the execution cost for each execution step under each execution plan based on the number of table tuples, column width, NULL record ratio, and characteristic values, such as distinct, MCV, and HB values, and certain cost calculation methods. The database then selects the execution plan that takes the lowest cost for the overall execution or for the return of the first tuple. These characteristic values are the statistics, which is the core for optimizing a query. Accurate statistics helps the analyzer select the most appropriate query plan. Generally, you can collect statistics of a table or that of some columns in a table using ANALYZE. You are advised to periodically execute ANALYZE or execute it immediately after you modified most contents in a table.

NOTE

DDL operations may cause statistics to change, resulting in plan changes. After DDL operations are performed on a table, check whether statistics need to be collected again.

Optimizing Queries Using GUC Parameters

Optimizing queries aims to select an efficient execution mode.

Take the following SQL statement as an example:

```
select count(1)
from customer inner join store_sales on (ss_customer_sk = c_customer_sk);
```

During execution of **customer inner join store_sales**, GaussDB supports nested loop, merge join, and hash join. The optimizer estimates the result set size and the execution cost of each join mode based on the statistics on the **customer** and **store_sales** tables. It then compares the costs and selects the one costing the least.

As mentioned above, the execution cost is calculated based on certain methods and statistics. If the actual execution cost cannot be accurately estimated, you need to optimize the execution plan by setting GUC parameters. For example, the **random_page_cost** parameter indicates the optimizer's calculation of the cost of a non-sequentially-fetched disk page. The default value is **4**. When the random

read speed of a machine disk (for example, an SSD) is high, you can decrease the value of this parameter. After the change, the cost of index scan is reduced, and the index scan mode is preferred when a plan is generated.

Optimizing Queries by Rewriting SQL Statements

Besides the preceding methods that improve the performance of the execution plan generated by the SQL engine, database administrators can also enhance SQL statement performance by rewriting SQL statements while retaining the original service logic based on the execution mechanism of the database and abundant practices.

This requires that database administrators know the customer services well and have professional knowledge of SQL statements. Below chapters will describe some common SQL rewriting scenarios.

6.2 Introduction to the SQL Execution Plan

6.2.1 Overview

The SQL execution plan is a node tree, which displays detailed procedure when GaussDB runs an SQL statement. A database operator indicates one step.

You can run the **EXPLAIN** command to view the execution plan generated for each query by an optimizer. The output of EXPLAIN has one row for each execution node, showing the basic node type and the cost estimation that the optimizer made for the execution of this node, as shown below.

```
gaussdb=# explain select * from t1,t2 where t1.c1=t2.c2;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=14.17..29.07 rows=20 width=180)
  Node/s: All datanodes
  -> Hash Join (cost=13.29..27.75 rows=20 width=180)
    Hash Cond: (t2.c2 = t1.c1)
    -> Streaming(type: REDISTRIBUTE) (cost=0.00..14.31 rows=20 width=104)
      Spawn on: All datanodes
      -> Seq Scan on t2 (cost=0.00..13.13 rows=20 width=104)
    -> Hash (cost=13.13..13.13 rows=21 width=76)
      -> Seq Scan on t1 (cost=0.00..13.13 rows=20 width=76)
(9 rows)
```

- Nodes at the bottom level are scan nodes. They scan tables and return raw rows. The types of scan nodes (sequential scans and index scans) vary depending on the table access methods. Objects scanned by the bottom layer nodes may not be row-store data (not directly read from a table), such as VALUES clauses and functions that return rows, which have their own types of scan nodes.
- If the query requires join, aggregation, sorting, or other operations on the raw rows, there will be other nodes above the scan nodes to perform these operations. In addition, there is more than one way to perform these operations, so different types of execution nodes may be displayed here.
- The first row (the upper-layer node) estimates the total execution cost of the execution plan. Such an estimate indicates the value that the optimizer tries to minimize.

Execution Plan Display Format

GaussDB provides four display formats: normal, pretty, summary, and run.

- normal: indicates that the default printing format is used.
- pretty: indicates that the new plan display format improved by GaussDB is used. The new format contains a plan node ID, directly and effectively analyzing performance.
- summary: indicates that the printing information analysis is added based on the pretty format.
- run: indicates that the information based on the summary format is exported as a CSV file for further analysis.

An example of an execution plan using the pretty format is as follows.

```
gaussdb=# explain select * from t1,t2 where t1.c1=t2.c2;
id |          operation          | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
 1 | -> Streaming (type: GATHER) |      20 |    180 | 29.07
 2 | -> Hash Join (3,5)         |      20 |    180 | 27.75
 3 | -> Streaming(type: REDISTRIBUTE) |     20 |    104 | 14.31
 4 | -> Seq Scan on t2         |      20 |    104 | 13.13
 5 | -> Hash                   |      21 |     76 | 13.13
 6 | -> Seq Scan on t1         |      20 |     76 | 13.13
(6 rows)

Predicate Information (identified by plan id)
-----
 2 --Hash Join (3,5)
    Hash Cond: (t2.c2 = t1.c1)
(2 rows)
```

You can change the display format of execution plans by setting the GUC parameter **explain_perf_mode**. Later examples use the pretty format by default.

Execution Plan Information

In addition to setting different display formats for an execution plan, you can use different EXPLAIN syntax to display execution plan information in detail. The following lists the common EXPLAIN syntax. For more EXPLAIN syntax, see [EXPLAIN](#).

- EXPLAIN *statement*: only generates an execution plan and does not execute. The *statement* indicates SQL statements.
- EXPLAIN ANALYZE *statement*: generates and executes an execution plan, and displays the execution summary. Then actual execution time statistics are added to the display, including the total elapsed time expended within each plan node (in milliseconds) and the total number of rows it actually returned.
- EXPLAIN PERFORMANCE *statement*: generates and executes the execution plan, and displays all execution information.

To measure the run time cost of each node in the execution plan, the current execution of EXPLAIN ANALYZE or EXPLAIN PERFORMANCE adds profiling overhead to query execution. Running EXPLAIN ANALYZE or EXPLAIN PERFORMANCE on a query sometimes takes longer time than executing the query normally. The amount of time that exceeds depends on the complexity of the query itself and the platform used.

Therefore, if an SQL statement is not finished after being running for a long time, run the **EXPLAIN** command to view the execution plan and then locate the fault. If the SQL statement has been properly executed, execute the EXPLAIN ANALYZE or EXPLAIN PERFORMANCE statement to check the execution plan and information to locate the fault.

6.2.2 Execution Plan Details

6.2.2.1 Execution Plans

The following SQL statement in pretty mode is used as an example:

```
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
gaussdb=# CREATE TABLE t1 (c1 int, c2 int);
gaussdb=# CREATE TABLE t2 (c1 int, c2 int);
gaussdb=# SET explain_perf_mode = pretty;
```

Run the **EXPLAIN** command and the output is as follows:

```
gaussdb=# EXPLAIN SELECT * FROM t1,t2 WHERE t1.c1 = t2.c2;
id |          operation          | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) |      20 |      16 | 28.69
2 | -> Hash Join (3,5)          |      20 |      16 | 27.75
3 | -> Streaming(type: REDISTRIBUTE) |      20 |      8 | 14.31
4 | -> Seq Scan on t2           |      20 |      8 | 13.13
5 | -> Hash                      |      21 |      8 | 13.13
6 | -> Seq Scan on t1           |      20 |      8 | 13.13
(6 rows)

Predicate Information (identified by plan id)
-----
2 --Hash Join (3,5)
   Hash Cond: (t2.c2 = t1.c1)
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE t1,t2;
```

- Interpretation of the execution plan column (horizontal):
 - **id**: execution operator node ID.
 - **operation**: name of an execution operator.
 - **E-rows**: number of output rows estimated by each operator.
 - **E-width**: estimated width of an output tuple of each operator.
 - **E-costs**: execution cost estimated by each operator.
 - **E-costs** is measured by the optimizer based on an overhead unit. Usually, fetching a disk page sequence is defined as a unit. Other overhead parameters are set based on the unit.
 - The overhead of each node (specified by **E-costs**) includes the overheads of all its child nodes.
 - Such an overhead reflects only what the optimizer is concerned about, but does not consider the time for passing result rows to the client. Although the time may play a very important role in the actual total time, it is ignored by the optimizer, because it cannot be changed by modifying the plan.

- Interpretation of the execution plan layers (vertical). The plan tree is from bottom to top. In pretty mode, operators at each layer are indented with ->. The bottom layer is generally a scan operator.
 - Operators id-4 and id-6 are the scan operators at the bottom layer.
 - i. Seq Scan on t1
The table scan operator scans the table **t1** using Seq Scan. At this layer, data in the table **t1** is read from a buffer or disk, and then transferred to the upper-layer node for hash join calculation.
 - ii. Seq Scan on t2
The table scan operator scans the table **t2** using Seq Scan. At this layer, data in the table **t2** is read from a buffer or disk, and then transferred to the upper-layer node for calculation.
 - Layer 2: **Hash**
The hash operator calculates the hash value of the operator transferred from the lower layer and generates a hash table for subsequent hash join operations.
 - Layer 2: Streaming(type: REDISTRIBUTE)
Streaming is a special operator that implements the core data shuffle function of the distributed architecture. Streaming (type: REDISTRIBUTE) redistributes data to all DNs based on the selected column. This operator redistributes data of **t2** to each DN based on the join column to prepare for t1 join. For details about the Streaming operator, see the subsequent operator description.
 - Layer 3: Hash Join (3,5)
The hash join operator is used to join data in the **t1** and **t2** tables in hash join mode and output the result data. **(3,5)** indicates that the result of the operator whose ID is 3 is joined with the result of the operator whose ID is 5.
 - Layer 4: Streaming (type: GATHER)
Streaming (type: GATHER) collects data from DNs for the CN.

6.2.2.2 Distributed Plan

Currently, the GaussDB optimizer can use three methods to develop statement execution policies in the distributed framework: generating a statement pushdown plan, a distributed execution plan, or a distributed execution plan for sending statements.

- A statement pushdown plan pushes query statements from a CN down to DNs for execution and returns the execution results to the CN.
- In a distributed execution plan, a CN compiles and optimizes query statements, generates a plan tree, and then sends the plan tree to DNs for execution. After the statements have been executed, execution results will be returned to the CN.
- A distributed execution plan for sending statements pushes queries that can be pushed down (mostly base table scanning statements) to DNs for execution. Then, the plan obtains the intermediate results and sends them to the CN, on which the remaining queries are to be executed.

Statement pushdown plan:

```
gaussdb=# CREATE TABLE t1(c1 int,c2 int);
gaussdb=# EXPLAIN SELECT * FROM t1;
          QUERY PLAN
-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
  Node/s: All datanodes
(2 rows)
```

Interpretation of plan columns:

- **Data Node Scan**
The SQL statement is pushed down to DNs for plan generation and execution.
- **Node/s: All datanodes**
The SQL statement will be executed on all DNs.

Run the **EXPLAIN VERBOSE** command to view the delivered SQL statements.

```
gaussdb=# EXPLAIN VERBOSE SELECT * FROM t1;
          QUERY PLAN
-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
  Output: t1.c1, t1.c2
  Node/s: All datanodes
  Remote query: SELECT c1, c2 FROM public.t1
(4 rows)
```

The execution plan contains the **Remote query** column, which describes the delivered SQL statement. To further view the plan after the SQL statement is pushed down to DNs, set the **max_datanode_for_plan** parameter. This parameter indicates the number of plans on DNs. The number of plans on DNs is determined by the smaller one between the number of DNs in the cluster and the value of this parameter. S

```
gaussdb=# SET max_datanode_for_plan = 1;
gaussdb=# EXPLAIN VERBOSE SELECT * FROM t1;
          QUERY PLAN
-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
  Output: t1.c1, t1.c2
  Node/s: All datanodes
  Remote query: SELECT c1, c2 FROM public.t1

Remote SQL: SELECT c1, c2 FROM public.t1
Datanode Name: datanode1
Seq Scan on public.t1 (cost=0.00..1.06 rows=6 width=8)
  Output: c1, c2
(10 rows)
```

The plan on datanode1 is displayed.

Distributed execution plan:

If an SQL statement cannot be pushed down, a distributed execution plan is generated first.

```
gaussdb=# EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.c1 = t1.c2;
id | operation | E-rows | E-memory | E-width | E-costs
-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 10 | | 16 | 2.79
2 | -> Nested Loop (3,5) | 10 | 1MB | 16 | 2.32
3 | -> Streaming (type: BROADCAST) | 2 | 2MB | 8 | 1.22
4 | -> Seq Scan on t1 | 1 | 1MB | 8 | 1.06
```

```

5 | -> Seq Scan on t2 | 10 | 1MB | 8 | 1.05
(5 rows)

Predicate Information (identified by plan id)
-----
4 --Seq Scan on t1
   Filter: (c1 = c2)
(2 rows)

===== Query Summary =====
-----
System available mem: 8193638KB
Query Max mem: 8280473KB
Query estimated mem: 5152KB
(3 rows)

```

A distributed execution plan contains a special streaming operator. It implements the core data shuffle function of the distributed architecture. Streaming has three types, which correspond to different data shuffle functions in the distributed architecture:

- Streaming (type: GATHER): The CN collects data from DNs.
- Streaming (type: REDISTRIBUTE): Data is redistributed to all the DNs based on selected columns.
- Streaming (type: BROADCAST): Data on the current DN is broadcast to other DNs.

Distributed execution plan for sending statements:

If neither of the two execution plans can be generated, an execution plan for sending statements is generated.

```

gaussdb=# EXPLAIN VERBOSE SELECT * FROM t1 limit 1;
          QUERY PLAN
-----
Limit (cost=0.00..0.00 rows=1 width=8)
 Output: t1.c1, t1.c2
 -> Data Node Scan on "__REMOTE_LIMIT_QUERY__" (cost=0.00..0.00 rows=1 width=8)
    Output: t1.c1, t1.c2
    Node/s: All datanodes
    Remote query: SELECT c1, c2 FROM ONLY public.t1 WHERE true LIMIT 1::bigint
(6 rows)

-- Drop the table.
gaussdb=# DROP TABLE t1;

```

A CN with the LIMIT statement cannot simply send statements to DNs and collect data, which is inconsistent with the semantics of the LIMIT statement. The execution plan distributes the LIMIT 1 statement to all DNs. After the intermediate result is collected to the CN, the CN performs LIMIT processing.

6.2.2.3 Execution Information

In SQL optimization process, you can use EXPLAIN ANALYZE or EXPLAIN PERFORMANCE to check the SQL statement execution information. By comparing estimation differences between actual implementation and the optimizer, basis for service optimization is provided. EXPLAIN PERFORMANCE provides the execution information on each DN, whereas EXPLAIN ANALYZE does not.

The execution result of the following SQL statement in pretty mode is used as an example:

```

-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE t1 (c1 int, c2 int);

```

```
gaussdb=# CREATE TABLE t2 (c1 int, c2 int);
gaussdb=# INSERT INTO t1 SELECT generate_series(1,10), generate_series(1,10);
gaussdb=# INSERT INTO t2 SELECT generate_series(1,10), generate_series(1,10);
gaussdb=# ANALYZE t1;
gaussdb=# ANALYZE t2;
gaussdb=# SELECT sum(t2.c1) FROM t1,t2 WHERE t1.c1 = t2.c2 GROUP BY t1.c2;
```

The output of running EXPLAIN PERFORMANCE is as follows:

```
gaussdb=# EXPLAIN PERFORMANCE SELECT sum(t2.c1) FROM t1,t2 WHERE t1.c1 = t2.c2 GROUP BY t1.c2;
id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-
memory | A-width | E-width | E-costs
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 64.022 | 10 | 10 | | 82KB | |
| | 16 | 3.31
2 | -> HashAggregate | [54.464,58.410] | 10 | 10 | | [27KB, 29KB] | |
16MB | [20,20] | 16 | 2.84
3 | -> Streaming(type: REDISTRIBUTE) | [54.385,58.318] | 10 | 10 | | [36KB, 36KB] | |
2MB | | 8 | 2.76
4 | -> Hash Join (5,7) | [0.471,0.906] | 10 | 10 | 5, 5 | [8KB, 8KB] | 1MB
| | 8 | 2.64
5 | -> Streaming(type: REDISTRIBUTE) | [0.053,0.385] | 10 | 10 | | [36KB, 36KB] | |
2MB | | 8 | 1.46
6 | -> Seq Scan on public.t2 | [0.014,0.021] | 10 | 10 | | [13KB, 13KB] | |
1MB | | 8 | 1.05
7 | -> Hash | [0.055,0.071] | 10 | 10 | | [293KB, 293KB] | 16MB
[24,24] | 8 | 1.05
8 | -> Seq Scan on public.t1 | [0.019,0.024] | 10 | 10 | | [13KB, 13KB] | |
1MB | | 8 | 1.05
(8 rows)
```

Predicate Information (identified by plan id)

```
-----+-----
4 --Hash Join (5,7)
Hash Cond: (t2.c2 = t1.c1)
(2 rows)
```

Memory Information (identified by plan id)

```
-----+-----
Coordinator Query Peak Memory:
Query Peak Memory: 1MB
DataNode Query Peak Memory
datanode1 Query Peak Memory: 6MB
datanode2 Query Peak Memory: 6MB
1 --Streaming (type: GATHER)
Peak Memory: 82KB, Estimate Memory: 64MB
2 --HashAggregate
datanode1 Peak Memory: 29KB, Estimate Memory: 16MB, Width: 20
datanode2 Peak Memory: 27KB, Estimate Memory: 16MB, Width: 20
3 --Streaming(type: REDISTRIBUTE)
datanode1 Peak Memory: 36KB, Estimate Memory: 2MB
datanode2 Peak Memory: 36KB, Estimate Memory: 2MB
datanode1 Stream Network: 1kB, Network Poll Time: 0.000; Data Deserialize Time: 0.000
datanode2 Stream Network: 1kB, Network Poll Time: 0.000; Data Deserialize Time: 0.000
4 --Hash Join (5,7)
datanode1 Peak Memory: 8KB, Estimate Memory: 1024KB
datanode2 Peak Memory: 8KB, Estimate Memory: 1024KB
5 --Streaming(type: REDISTRIBUTE)
datanode1 Peak Memory: 36KB, Estimate Memory: 2MB
datanode2 Peak Memory: 36KB, Estimate Memory: 2MB
datanode1 Stream Network: 1kB, Network Poll Time: 0.000; Data Deserialize Time: 0.000
datanode2 Stream Network: 1kB, Network Poll Time: 0.000; Data Deserialize Time: 0.000
6 --Seq Scan on public.t2
datanode1 Peak Memory: 13KB, Estimate Memory: 1024KB
datanode2 Peak Memory: 13KB, Estimate Memory: 1024KB
7 --Hash
datanode1 Peak Memory: 293KB, Estimate Memory: 16MB, Width: 24
datanode2 Peak Memory: 293KB, Estimate Memory: 16MB, Width: 24
datanode1 Buckets: 32768 Batches: 1 Memory Usage: 1kB
```

```

    datanode2 Buckets: 32768 Batches: 1 Memory Usage: 1kB
8 --Seq Scan on public.t1
    datanode1 Peak Memory: 13KB, Estimate Memory: 1024KB
    datanode2 Peak Memory: 13KB, Estimate Memory: 1024KB
(34 rows)

-----
Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
    Output: (sum(t2.c1)), t1.c2
    Node/s: All datanodes
2 --HashAggregate
    Output: sum(t2.c1), t1.c2
    Group By Key: t1.c2
3 --Streaming(type: REDISTRIBUTE)
    Output: t1.c2, t2.c1
    Distribute Key: t1.c2
    Spawn on: All datanodes
    Consumer Nodes: All datanodes
4 --Hash Join (5,7)
    Output: t1.c2, t2.c1
5 --Streaming(type: REDISTRIBUTE)
    Output: t2.c1, t2.c2
    Distribute Key: t2.c2
    Spawn on: All datanodes
    Consumer Nodes: All datanodes
6 --Seq Scan on public.t2
    Output: t2.c1, t2.c2
    Distribute Key: t2.c1
7 --Hash
    Output: t1.c2, t1.c1
8 --Seq Scan on public.t1
    Output: t1.c2, t1.c1
    Distribute Key: t1.c1
(26 rows)

-----
Datanode Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
    (actual time=57.765..64.022 rows=10 loops=1)
    (Buffers: 0)
    (CPU: ex c/r=16607284, ex row=10, ex c/c=166072848, inc c/c=166072848)
2 --HashAggregate
    datanode1 (actual time=54.460..54.464 rows=6 loops=1)
    datanode2 (actual time=58.407..58.410 rows=4 loops=1)
    datanode1 (Buffers: 0)
    datanode2 (Buffers: 0)
    datanode1 (CPU: ex c/r=34274, ex row=6, ex c/c=205644, inc c/c=141279464)
    datanode2 (CPU: ex c/r=59030, ex row=4, ex c/c=236120, inc c/c=151512902)
3 --Streaming(type: REDISTRIBUTE)
    datanode1 (actual time=19.021..54.385 rows=6 loops=1)
    datanode2 (actual time=14.622..58.318 rows=4 loops=1)
    datanode1 (Buffers: 0)
    datanode2 (Buffers: 0)
    datanode1 (CPU: ex c/r=23512303, ex row=6, ex c/c=141073820, inc c/c=141073820)
    datanode2 (CPU: ex c/r=37819195, ex row=4, ex c/c=151276782, inc c/c=151276782)
4 --Hash Join (5,7)
    datanode1 (actual time=0.451..0.471 rows=6 loops=1)
    datanode2 (actual time=0.869..0.906 rows=4 loops=1)
    datanode1 (Buffers: shared hit=1)
    datanode2 (Buffers: shared hit=1)
    datanode1 (CPU: ex c/r=78319, ex row=12, ex c/c=939828, inc c/c=1220560)
    datanode2 (CPU: ex c/r=145606, ex row=8, ex c/c=1164852, inc c/c=2347916)
5 --Streaming(type: REDISTRIBUTE)
    datanode1 (actual time=0.046..0.053 rows=6 loops=1)
    datanode2 (actual time=0.358..0.385 rows=4 loops=1)
    datanode1 (Buffers: 0)
    datanode2 (Buffers: 0)
    datanode1 (CPU: ex c/r=22796, ex row=6, ex c/c=136780, inc c/c=136780)

```

```

datanode2 (CPU: ex c/r=249361, ex row=4, ex cyc=997444, inc cyc=997444)
6 --Seq Scan on public.t2
  datanode1 (actual time=0.016..0.021 rows=6 loops=1)
  datanode2 (actual time=0.010..0.014 rows=4 loops=1)
  datanode1 (Buffers: shared hit=1)
  datanode2 (Buffers: shared hit=1)
  datanode1 (CPU: ex c/r=8410, ex row=6, ex cyc=50460, inc cyc=50460)
  datanode2 (CPU: ex c/r=8769, ex row=4, ex cyc=35076, inc cyc=35076)
7 --Hash
  datanode1 (actual time=0.055..0.055 rows=6 loops=1)
  datanode2 (actual time=0.071..0.071 rows=4 loops=1)
  datanode1 (Buffers: shared hit=1)
  datanode2 (Buffers: shared hit=1)
  datanode1 (CPU: ex c/r=16183, ex row=6, ex cyc=97100, inc cyc=143952)
  datanode2 (CPU: ex c/r=32248, ex row=4, ex cyc=128992, inc cyc=185620)
8 --Seq Scan on public.t1
  datanode1 (actual time=0.014..0.019 rows=6 loops=1)
  datanode2 (actual time=0.018..0.024 rows=4 loops=1)
  datanode1 (Buffers: shared hit=1)
  datanode2 (Buffers: shared hit=1)
  datanode1 (CPU: ex c/r=7808, ex row=6, ex cyc=46852, inc cyc=46852)
  datanode2 (CPU: ex c/r=14157, ex row=4, ex cyc=56628, inc cyc=56628)
(53 rows)

```

User Define Profiling

```

-----
Segment Id: 2 Track name: Datanode build connection
  datanode1 (time=0.117 total_calls=1 loops=1)
  datanode2 (time=0.161 total_calls=1 loops=1)
Plan Node id: 1 Track name: coordinator get datanode connection
  coordinator1: (time=0.023 total_calls=1 loops=1)
Plan Node id: 1 Track name: Coordinator check and update node definition
  coordinator1: (time=0.001 total_calls=1 loops=1)
Plan Node id: 1 Track name: Coordinator serialize plan
  coordinator1: (time=1.289 total_calls=1 loops=1)
Plan Node id: 1 Track name: Coordinator send query id with sync
  coordinator1: (time=0.091 total_calls=1 loops=1)
Plan Node id: 1 Track name: Coordinator send begin command
  coordinator1: (time=0.002 total_calls=1 loops=1)
Plan Node id: 1 Track name: Coordinator start transaction and send query
  coordinator1: (time=0.044 total_calls=1 loops=1)
Plan Node id: 2 Track name: Datanode start up stream thread
  datanode1 (time=0.172 total_calls=1 loops=1)
  datanode2 (time=0.173 total_calls=1 loops=1)
(18 rows)

```

==== Query Summary =====

```

-----
Datanode executor start time [datanode1, datanode2]: [0.451 ms,0.475 ms]
Datanode executor run time [datanode1, datanode2]: [54.512 ms,58.450 ms]
Datanode executor end time [datanode2, datanode1]: [0.098 ms,0.105 ms]
Remote query poll time: 0.000 ms, Deserialize time: 0.000 ms
System available mem: 8190361KB
Query Max mem: 8280473KB
Query estimated mem: 6464KB
Coordinator executor start time: 0.285 ms
Coordinator executor run time: 64.083 ms
Coordinator executor end time: 0.117 ms
Total network : 1kB
Planner runtime: 1.242 ms
Plan size: 4251 byte
Query Id: 72620543991481051
Total runtime: 64.786 ms
(15 rows)

-- Drop the table.
gaussdb=# DROP TABLE t1,t2;

```

In the preceding example, the execution information consists of the following parts:

1. The plan is displayed as a table, which contains 11 columns: **id**, **operation**, **A-time**, **A-rows**, **E-rows**, **E-distinct**, **Peak Memory**, **E-memory**, **A-width**, **E-width**, and **E-costs**. The meanings of the plan-type columns (**id**, **operation**, or columns started with **E**) are the same as those when EXPLAIN is executed. For details, see [Execution Plans](#). The definition of **A-time**, **A-rows**, **E-distinct**, **Peak Memory**, and **A-width** are described as follows:
 - **A-time**: execution completion time of the current operator. The values in the brackets indicate the minimum and maximum DN completion time.
 - **A-rows**: number of actual output tuples of the operator
 - **E-distinct**: estimated distinct values at both ends of the hash join operator.
 - **Peak Memory**: peak memory used by the operator during execution.
 - **A-width**: actual tuple width in each row of the current operator. This parameter is valid only for heavy memory operators, including (Vec)HashJoin, (Vec)HashAgg, (Vec)HashSetOp, (Vec)Sort, and (Vec)Materialize. The (Vec)HashJoin calculation width is the width of its right subtree operator and will be displayed on the right subtree.
2. **Predicate Information (identified by plan id)**:

This part displays the static information that does not change in the plan execution process, such as some join conditions and filter information.

In this example:

 - In **4 --Hash Join (5,7)**, **4** indicates the operator whose ID is 4, and **(5,7)** indicates that the operator whose ID is 5 and the operator whose ID is 7 are joined.
 - **Hash Cond: (t2.c2 = t1.c1)** indicates the hash join condition.
3. **Memory Information (identified by plan id)**:

This part displays the memory usage information printed by certain operators (mainly Hash and Sort), including **peak memory**, **control memory**, **estimate memory**, **width**, **auto spread num**, and **early spilled**; and spill details, including **spill Time(s)**, **inner/outer partition spill num**, **temp file num**, spilled data volume, and **written disk IO** [*min*, *max*].
4. **Targetlist Information (identified by plan id)**:

This part displays the target columns provided by each operator.
5. **DataNode Information (identified by plan id)**:

The execution time, CPU, and buffer usage of each operator are printed in this part.
6. **User Define Profiling**:

This part displays the thread interaction between CNs and DNs in the stream plan.
7. **=====
Query Summary
=====**:

This part displays the total execution time and network traffic, including the maximum and minimum execution time in the initialization and end phases, available system memory when the current statement is executed, and estimated statement memory.

6.2.3 Operators

6.2.3.1 Keyword Overview

Table access modes

- Seq Scan
Scans all rows of the table in sequence.
- Index Scan
Scans indexes. The optimizer uses a two-step plan: the child plan node visits an index to find the locations of rows matching the index condition, and then the upper plan node actually fetches those rows from the table itself. Fetching rows separately is much more expensive than reading them sequentially, but because not all pages of the table have to be visited, this is still cheaper than a sequential scan. The upper-layer planning node sorts index-identified rows based on their physical locations before reading them. This minimizes the independent capturing overhead.
If there are separate indexes on multiple columns referenced in WHERE, the optimizer might choose to use an AND or OR combination of the indexes. However, this requires the visiting of both indexes, so it is not necessarily a win compared to using just one index and treating the other condition as a filter.
The following index scans featured with different sorting mechanisms are involved:
 - Bitmap index scan
Fetches data pages using a bitmap.
 - Index scan using index_name
Uses simple index search, which fetches data from an index table in the sequence of index keys. This mode is commonly used when only a small amount of data needs to be fetched from a large data table or when the ORDER BY condition is used to match the index sequence to reduce the sorting time.
 - Index-Only Scan
Scans the index that contains all required data, instead of referencing a table.
- Bitmap Heap Scan
Reads pages from bitmaps created by other operations and filters out the rows that do not meet the conditions. Bitmap heap scan can avoid random I/Os and accelerate read speed.
- TID Scan
Scans a table by a tuple ID.
- Index Ctid Scan
Scans a table based on the CTID index.
- CTE Scan
Specifies that CTE evaluates subquery operations and stores query results as a temporary table. The temporary table is scanned by the CTE Scan operator.

- Foreign Scan
Reads data from a remote data source.
- Function Scan
Obtains result sets returned by functions and returns them as the rows read from tables.
- Sample Scan
Queries and returns sampled data.
- Subquery Scan
Reads subquery results.
- Values Scan
Reads constants as part of the **VALUES** command.
- WorkTable Scan
Scans a work table. Data is read in the middle of an operation which is usually a recursive operation declared using WITH RECURSIVE.

Table join modes

- Nested Loop
A nested loop is used for queries that have a smaller dataset connected. In a nested loop join, the outer table drives the inner table and each row returned from the outer table should have a matching row in the inner table. The returned result set of all queries should be less than 10,000. The table that returns a smaller subset will work as an outer table, and indexes are recommended for connection columns of the inner table.
- (Sonic) Hash Join
A hash join is used for large tables. The optimizer uses a hash join, in which rows of one table are entered into an in-memory hash table, after which the other table is scanned and the hash table is probed for matches to each row. Sonic and non-Sonic hash joins differ in their hash table structures, which do not affect the execution result set.
- Merge Join
In most cases, the execution performance of a merge join is lower than that of a hash join. However, if the source data has been pre-sorted and no more sorting is required during the merge join, its performance excels.

Operators

- sort
Sorts the result set.
- filter
The EXPLAIN output shows the WHERE clause being applied as a filter condition attached to the Seq Scan plan node. This means that the plan node checks the condition for each row it scans, and returns only the ones that meet the condition. The estimated number of output rows has been reduced because of the WHERE clause. However, the scan will still have to visit all 10,000 rows, as a result, the cost is not decreased. It increases a bit (by 10,000 x **cpu_operator_cost**) to reflect the extra CPU time spent on checking the WHERE condition.

- **LIMIT**
Limits the number of output execution results. If a LIMIT condition is added, not all rows are retrieved.
- **Append**
Appends sub-operation results.
- **Aggregate**
Aggregates the results generated from querying rows. It can be an aggregation of statements such as GROUP BY, UNION, and SELECT DISTINCT.
- **BitmapAnd**
Specifies the AND operation of a bitmap, which is used to form a bitmap that matches more complex conditions.
- **BitmapOr**
Specifies the OR operation of a bitmap, which is used to form a bitmap that matches more complex conditions.
- **Gather**
Gathers data of parallel threads.
- **Group**
Groups rows to perform the GROUP BY operation.
- **GroupAggregate**
Aggregates the pre-sorted rows of the GROUP BY operation.
- **Hash**
Hashes rows for the parent query. It is usually used to perform the JOIN operation.
- **HashAggregate**
Aggregates the result rows of GROUP BY by using a hash table.
- **Merge Append**
Merges subquery results in a way that preserves the sort order. It can be used to merge sorted rows in a table partition.
- **ProjectSet**
Executes a function on the returned result set.
- **Recursive Union**
Performs a union operation on all steps of a recursive function.
- **SetOp**
Specifies a set operation, such as INTERSECT or EXCEPT.
- **Unique**
Removes duplicates from an ordered result set.
- **HashSetOp**
Specifies a strategy for set operations such as INTERSECT or EXCEPT. It uses Append to avoid pre-sorted input.
- **LockRows**
Locks problematic rows to prevent other queries from writing, but allows reading.

- **Materialize**
Stores subquery results in the memory so that the parent query can quickly access and obtain the subquery results.
- **Result**
Returns a value without scanning.
- **WindowAgg**
Specifies a window aggregate function, which is generally triggered by the OVER statement.
- **Merge**
Performs a merge operation.
- **StartWith Operator**
Specifies the hierarchical query operator, which is used to perform recursive query operations.
- **Rownum**
Filters the row number in the query result. It usually appears in the ROWNUM clause.
- **Index Cond**
Specifies the index scan conditions.
- **Unpivot**
Specifies a transpose operator.

Partition pruning

- **Partition Iterator**
Partition iterator, which usually indicates that a subquery is an operation on a partition.
- **Iterations**
Specifies the number of iterations performed by the partition iteration operator on level-1 partitions. If **PART** is displayed, dynamic pruning is used. For example, **Iterations: 4** indicates that the iteration operator needs to traverse four level-1 partitions. **Iterations: PART** indicates that the number of level-1 partitions to be traversed is determined by parameter conditions of the partition key.
- **Selected Partitions**
Specifies the selected level-1 partitions for pruning. **m..n** indicates that partitions **m** to **n** are selected. Multiple consecutive partitions are separated by commas (.). For example, **Selected Partitions: 2..4,7** indicates that partitions 2, 3, 4, and 7 are selected.
- **Sub Iterations**
Specifies the number of iterations performed by the partition iteration operator on level-2 partitions. If **PART** is displayed, dynamic pruning is used. For example, **Sub Iterations: 4** indicates that the iteration operator needs to traverse four level-2 partitions. **Iterations: PART** indicates that the number of level-2 partitions to be traversed is determined by parameter conditions of the partition key.

- **Selected Subpartitions**
Specifies the selected level-2 partitions for pruning, which is in the format like: level-1 partition number:level-2 partition number.
For example, **Selected Subpartitions: 2:1 3:2** indicates that level-2 partition 1 of the second level-1 partition and level-2 partition 2 of the third level-1 partition are selected. **Selected Subpartitions: ALL** indicates that all level-2 partitions are selected.

Distributed Keywords

- **Data Node Scan**
Performs plan generation and execution on DNs.
- **Node/s: All datanodes**
Specifies the nodes to which the plan is delivered.
- **Remote query**
Specifies the delivered SQL statement.
- **Streaming**
Specifies streaming operators, including gather, redistribute, and broadcast.

Other keywords

- **InitPlan**
Indicates a non-related subplan.
- **SubPlan**
Correlated subplan.

6.2.3.2 Table Access Modes

This section describes how to disable the function of pushing down operators to DNs to facilitate operator display and description.

```
gaussdb=# SET enable_fast_query_shipping = off;
```

6.2.3.2.1 Seq Scan

Description

The Seq scan operator is a universal one. It scans a table in a certain direction (forward or backward) and returns all rows that meet the filter criteria.

Typical Scenarios

- The table has no index and needs to be scanned.
- The table has indexes, but most of the data in the table needs to be scanned.

Examples

Example 1: The table has no index and needs to be scanned.

```
-- Prepare data.  
gaussdb=# DROP TABLE IF EXISTS t1;
```

```

gaussdb=# CREATE TABLE t1 (c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100), 2, 3);
INSERT 0 100

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 = 2;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.06..13.26 rows=1 width=96)
  Node/s: datanode1
  -> Seq Scan on t1 (cost=0.00..13.16 rows=1 width=96)
      Filter: (c1 = 2::numeric)
(4 rows)

-- Drop the table.
gaussdb=# DROP TABLE t1;

```

In the preceding example, the output of the Seq scan operator is described in [Table 6-2](#).

Table 6-2 Output information of the Seq scan operator

Item	Description
Seq Scan	Operator name.
Filter	Filter predicate of the operator. In the example, the filter condition is the value of column c1 is 2 . When a query is executed, rows that meet these conditions are included in the final result set.

Example 2: The table has indexes, but most of the data in the table needs to be scanned.

```

-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# CREATE INDEX idx_c1 on t1(c1);
CREATE INDEX
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 100000), 2, 3);
INSERT 0 100000

-- Collect statistics.
gaussdb=# ANALYZE t1;

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 <= 80000;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=4.00..4698.38 rows=80136 width=16)
  Node/s: All datanodes
  -> Seq Scan on t1 (cost=0.00..942.00 rows=80136 width=16)
      Filter: (c1 <= 80000::numeric)
(4 rows)

-- Drop the table.
gaussdb=# DROP TABLE t1;

```

In the preceding example, the output of the Seq scan operator is described in [Table 6-3](#).

Table 6-3 Output information of the Seq scan operator

Item	Description
Seq Scan	Operator name.
Filter	Filter predicate of the operator. In the example, the filter condition is the value of column c1 is less than or equal to 80000. When a query is executed, rows that meet these conditions are included in the final result set.

6.2.3.2.2 Index Scan

Description

In an index scan, the database uses the index column specified by a statement to retrieve rows by traversing the index tree. When the database scans an index for a value, the value to be searched for can be found after n I/Os occur, where n is the height of the B-tree index. Generally, index scan is used to retrieve table data. The database reads the index block sequentially, finds the corresponding index key value, and then reads the corresponding table tuple based on the TID corresponding to the index key. When the data volume is large but the query result set is small, the index scan efficiency is higher than the Seq Scan efficiency.

Typical Scenarios

- Querying specific rows in a table: When a query statement contains a WHERE clause whose conditions can be matched by index columns, GaussDB uses the index scan to search for rows that meet the conditions.
- Sorting: When a query statement contains the ORDER BY clause whose columns can be sorted by index, GaussDB uses the index scan for sorting.
- Aggregating: When a query statement contains the GROUP BY clause whose columns can be grouped by index, GaussDB uses the index scan for aggregation.
- Connecting: When a query statement contains a JOIN operation whose columns can be matched by index, GaussDB uses the index scan to perform the join operation.

Examples

Example 1: Conditions in the WHERE clause can be matched by index columns.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS test;
gaussdb=# CREATE TABLE test (c1 int, c2 int);
CREATE TABLE
```

```

gaussdb=# CREATE INDEX c1_idx ON test (c1);
CREATE INDEX
gaussdb=# INSERT INTO test SELECT generate_series(1, 1000000), random()::integer;
INSERT 0 1000000
gaussdb=# INSERT INTO test SELECT generate_series(1, 1000000), random()::integer;
INSERT 0 1000000
gaussdb=# INSERT INTO test SELECT generate_series(1, 1000000), random()::integer;
INSERT 0 1000000

-- Collect statistics.
gaussdb=# ANALYZE test;
gaussdb=# EXPLAIN SELECT /*+ indexscan(test c1_idx) */ * FROM test WHERE c1 > 990000;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=4.00..25494.30 rows=31127 width=8)
  Node/s: All datanodes
  -> Index Scan using c1_idx on test (cost=0.00..24035.17 rows=31127 width=8)
      Index Cond: (c1 > 990000)
(4 rows)

```

In the preceding example, the output of the index scan operator is as follows.

Item	Description
Index Scan	Operator name.
Index Cond	Filter predicate of the operator. In the example, the filter condition is the value of column c1 is greater than 990000. When a query is executed, rows that meet these conditions are included in the final result set.

Example 2: Columns in the ORDER BY clause can be sorted by index.

```

gaussdb=# EXPLAIN SELECT /*+ indexscan(test c1_idx) */ * FROM test ORDER BY c1;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=4.00..190456.74 rows=3000000 width=8)
  Merge Sort Key: c1
  Node/s: All datanodes
  -> Index Scan using c1_idx on test (cost=0.00..49831.74 rows=3000000 width=8)
(4 rows)

```

Example 3: Columns in the GROUP BY clause can be grouped by index.

```

gaussdb=# EXPLAIN SELECT /*+ indexscan(test c1_idx) */ c1 FROM test GROUP BY c1;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=4.00..150955.36 rows=2077304 width=4)
  Node/s: All datanodes
  -> Group (cost=0.00..53581.74 rows=2077304 width=4)
      Group By Key: c1
      -> Index Only Scan using c1_idx on test (cost=0.00..49831.74 rows=3000000 width=4)
(5 rows)

-- Drop.
gaussdb=# DROP TABLE test;

```

6.2.3.2.3 Index Only Scan

Description

Index-only scan is a query optimization technology in GaussDB. It can improve query performance by scanning only indexes without accessing table data. If the query condition involves only an index column of a table, you can use index-only scan to optimize the query. Index-only scan directly scans indexes, reducing I/O operations and CPU overhead and improving query performance.

Typical Scenarios

You only need to query the value of the index column without accessing other columns in the table. For example, query the maximum or minimum value of a column in a table, or query the number of different values in a column.

Examples

Example: The target column contains only index columns.

```
-- Prepare data.
gaussdb=# CREATE TABLE test2 (a int, b int);
CREATE TABLE
gaussdb=# CREATE INDEX test2_idx ON test2 (a, b);
CREATE INDEX

-- Randomly insert 1,000 data records.
gaussdb=# INSERT INTO test2 VALUES(generate_series(1, 1000), generate_series(1, 1000));
INSERT 0 1000

-- Collect statistics.
gaussdb=# ANALYZE test2;

-- Execution result.
gaussdb=# EXPLAIN SELECT a, b FROM test2 WHERE a = 10 AND b = 20;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.06..8.36 rows=1 width=8)
Node/s: datanode2
-> Index Only Scan using test2_idx on test2 (cost=0.00..8.27 rows=1 width=8)
    Index Cond: ((a = 10) AND (b = 20))
(4 rows)

-- Drop.
gaussdb=# DROP TABLE test2;
```

In the preceding example, the output of the index-only scan operator is as follows.

Item	Description
Index Only Scan	Operator name.
Index Cond	Filter predicate of the operator. In the example, the filter condition is that the value of column a is equal to 10 and the value of column b is equal to 20 . When a query is executed, rows that meet these conditions are included in the final result set.

6.2.3.2.4 Bitmap

Description

1. Bitmap index scan: This operator uses the index scan to form a bitmap based on the TIDs of tuples that meet the filter conditions and returns the bitmap to the upper layer. The BITMAPAND or BITMAPOR operation can be performed among multiple result bitmaps scanned by multiple tables. Compared with the index scan, the BITMAPAND or BITMAPOR operation needs to compare the tuples filtered between tables, reducing filtering time.
2. Bitmap heap scan: The lower layer of this operator contains the bitmap index scan, which is used to scan the bitmap returned by the lower-layer operator to determine which tuples meet the certain conditions, obtain the tuples, and return them to the upper layer. The optimizer uses the bitmap index scan based on cost estimation and needs to return tuples that meet certain conditions. In this case, the upper layer usually uses the bitmap heap scan to filter tuples by scanning bitmaps and return the tuples.
3. BitmapOr: This operator performs the OR operation on the lower-layer bitmap and returns the bitmap after the operation.
4. BitmapAnd: This operator performs the AND operation on the lower-layer bitmap and returns the bitmap after the operation.

The prerequisite for using bitmap index scan and bitmap heap scan is that the GUC parameter **enable_bitmapscan** has been set to **on**.

Typical Scenarios

Multiple tables are associated, indexes are available on the associated columns of one or more tables, and there is enough data in some or all tables.

Examples

Example: Join multiple tables with indexes.

```
-- Prepare data. Create three tables with data of the numeric type, randomly fill in data, and create indexes
for each attribute of each table.
gaussdb=# SET enable_default_ustore_table = on;
SET
gaussdb=# CREATE TABLE t_btm_test_1(a numeric,b numeric,c numeric);
CREATE TABLE
gaussdb=# CREATE TABLE t_btm_test_2(a numeric,b numeric,c numeric);
CREATE TABLE
gaussdb=# CREATE TABLE t_btm_test_3(a numeric,b numeric,c numeric);
CREATE TABLE
gaussdb=# DECLARE
n1 numeric := 100;
n2 numeric := 0;
n3 numeric := 100;
BEGIN
WHILE n1 > 0 LOOP
WHILE n2 < 100 LOOP
WHILE n3 > 0 LOOP
INSERT INTO t_btm_test_1 VALUES(n1, n2, n3);
INSERT INTO t_btm_test_2 VALUES(n1, n3, n2);
INSERT INTO t_btm_test_3 VALUES(n2, n1, n3);
n3 := n3 - 1;
END LOOP;
n2 := n2 + 1;
END LOOP;
```

```
n1 := n1 - 1;
END LOOP;
END;
/
ANONYMOUS BLOCK EXECUTE

gaussdb=# CREATE INDEX idx_btm_test_11 ON t_btm_test_1 USING UBTREE (a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_12 on t_btm_test_1 USING UBTREE (b);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_13 on t_btm_test_1 USING UBTREE (c);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_21 on t_btm_test_2 USING UBTREE(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_22 on t_btm_test_2 USING UBTREE (b);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_23 on t_btm_test_2 USING UBTREE (c);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_31 on t_btm_test_3 USING UBTREE (a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_32 on t_btm_test_3 USING UBTREE (b);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_33 on t_btm_test_3 USING UBTREE (c);
CREATE INDEX

-- Collect statistics.
gaussdb=# ANALYZE t_btm_test_1;
gaussdb=# ANALYZE t_btm_test_2;
gaussdb=# ANALYZE t_btm_test_3;

-- Execution result.
gaussdb=# SET enable_bitmapsca = on;
SET
gaussdb=# EXPLAIN SELECT /*+ hashjoin(t_btm_test_1 t_btm_test_2 t_btm_test_3)
BitmapScan(t_btm_test_2 idx_btm_test_21 idx_btm_test_22) BitmapScan(t_btm_test_3 idx_btm_test_31)*/ *
FROM t_btm_test_1 , t_btm_test_2 , t_btm_test_3
WHERE t_btm_test_1.a = t_btm_test_2.b AND
t_btm_test_2.a = t_btm_test_3.c AND
t_btm_test_3.a = t_btm_test_1.c AND
t_btm_test_1.a = 1
AND (
(t_btm_test_2.a = 1 AND t_btm_test_3.a = 1) OR
(t_btm_test_2.a = 3 AND t_btm_test_3.a = 3)
);

```

QUERY PLAN

```
Streaming (type: GATHER) (cost=3632.18..5782.71 rows=735 width=42)
Node/s: All datanodes
-> Hash Join (cost=3628.18..5748.21 rows=735 width=42)
Hash Cond: (t_btm_test_1.c = t_btm_test_3.a)
-> Bitmap Heap Scan on t_btm_test_1 (cost=123.65..2227.20 rows=10169 width=14)
Recheck Cond: (a = 1::numeric)
-> Bitmap Index Scan on idx_btm_test_11 (cost=0.00..122.38 rows=10169 width=0)
Index Cond: (a = 1::numeric)
-> Hash (cost=3504.44..3504.44 rows=14 width=28)
-> Streaming(type: BROADCAST) (cost=916.01..3504.44 rows=14 width=28)
Spawn on: All datanodes
-> Hash Join (cost=916.01..3503.81 rows=7 width=28)
Hash Cond: (t_btm_test_3.c = t_btm_test_2.a)
Join Filter: (((t_btm_test_2.a = 1::numeric) AND (t_btm_test_3.a = 1::numeric)) OR
((t_btm_test_2.a = 3::numeric) AND (t_btm_test_3.a = 3::numeric)))
-> Bitmap Heap Scan on t_btm_test_3 (cost=244.56..2436.73 rows=19921 width=14)
Recheck Cond: ((a = 1::numeric) OR (a = 3::numeric))
-> BitmapOr (cost=244.56..244.56 rows=20022 width=0)
-> Bitmap Index Scan on idx_btm_test_31 (cost=0.00..122.06 rows=10081
width=0)
Index Cond: (a = 1::numeric)
-> Bitmap Index Scan on idx_btm_test_31 (cost=0.00..117.53 rows=9941
```

```

width=0)
      Index Cond: (a = 3::numeric)
-> Hash (cost=669.12..669.12 rows=372 width=14)
  -> Streaming(type: BROADCAST) (cost=346.49..669.12 rows=372 width=14)
      Spawn on: All datanodes
  -> Bitmap Heap Scan on t_btm_test_2 (cost=346.49..654.48 rows=186 width=14)
      Recheck Cond: ((b = 1::numeric) AND ((a = 1::numeric) OR (a = 3::numeric)))
  -> BitmapAnd (cost=346.49..346.49 rows=187 width=0)
      -> Bitmap Index Scan on idx_btm_test_22 (cost=0.00..111.80 rows=9481
width=0)
      Index Cond: (b = 1::numeric)
  -> BitmapOr (cost=234.42..234.42 rows=19702 width=0)
      -> Bitmap Index Scan on idx_btm_test_21 (cost=0.00..117.00
rows=9801 width=0)
      Index Cond: (a = 1::numeric)
  -> Bitmap Index Scan on idx_btm_test_21 (cost=0.00..117.38
rows=9901 width=0)
      Index Cond: (a = 3::numeric)
(34 rows)
-- Drop the table.
gaussdb=# DROP TABLE t_btm_test_1, t_btm_test_2, t_btm_test_3;

```

In the preceding example, the output of the bitmap heap scan operator is as follows.

Item	Description
Bitmap Heap Scan	Operator name.
Recheck Cond	After the index scan is performed, the database checks the filter predicate again. In the example, the filter condition of Recheck Cond: ((a = 1::numeric) OR (a = 3::numeric)) is that the value of column a is equal to 1 or 3 . When a query is executed, rows that meet these conditions are included in the final result set.

In the preceding example, the output of the bitmap index scan operator is as follows.

Item	Description
Bitmap Index Scan	Operator name.
Index Cond	Filter predicate of the operator. When a query is executed, rows that meet these conditions are included in the final result set.

In the preceding example, the output of the BitmapOr and BitmapAnd operators is as follows.

Item	Description
BitmapOr	Operator name, indicating that the OR operation is performed on the bitmap returned by the lower layer to form a bitmap that matches more complex conditions.
BitmapAnd	Operator name, indicating that the AND operation is performed on the bitmap returned by the lower layer to form a bitmap that matches more complex conditions.

6.2.3.2.5 TID Scan

Description

This is a row number scan operator, which uses the row number (**ctid**) to filter and return the tuples. In the Astore scenario, data is stored in the heap page by row. In addition to the value of a storage column, the corresponding row number is also stored in the B-tree index. Therefore, GaussDB supports quick retrieval by row number.

The row number is in the following format:

```
(page_number, item_number) -- page_number starts from 0, and the value of item_number starts from 1.
```

The corresponding tuple in the disk or cache is found based on the given **ctid** value.

Typical Scenarios

Prerequisites: The GUC parameter **enable_tidscan** has been set to **on**.

You can quickly search for data by specifying a valid row number if a table supports TID scan. Currently, GaussDB selects TID scan only when the WHERE condition is =. IN is not supported.

Examples

```
-- Prepare data.
gaussdb=# CREATE TABLE t_tid_test(a numeric,b numeric,c numeric);
CREATE TABLE
gaussdb=# DECLARE
n1 numeric := 10;
n2 numeric := 0;
n3 numeric := 100;
BEGIN
WHILE n1 > 0 LOOP
WHILE n2 < 100 LOOP
WHILE n3 > 0 LOOP
INSERT INTO t_tid_test VALUES(n1, n2, n3);
n3 := n3 - 1;
END LOOP;
END LOOP;
n2 := n2 + 1;
```

```

        END LOOP;
        n1 := n1 - 1;
    END LOOP;
END;
/
ANONYMOUS BLOCK EXECUTE

-- Collect statistics.
gaussdb=# ANALYZE t_tid_test;

-- Execution result.
gaussdb=# SET enable_tidscan = on;
SET
gaussdb=# EXPLAIN ANALYZE SELECT * FROM t_tid_test WHERE ctid = '(0,10)::tid;
                QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.07..4.11 rows=1 width=14) (actual time=3.089..3.282 rows=2 loops=1)
  Node/s: All datanodes
  -> Tid Scan on t_tid_test (cost=0.00..4.01 rows=1 width=14) (actual time=[0.051,0.052]..[0.055,0.056],
rows=2)
    TID Cond: (ctid = '(0,10)::tid)
  Total runtime: 3.473 ms
(5 rows)

-- Drop the table.
gaussdb=# DROP TABLE t_tid_test;
    
```

In the preceding example, the output of the TID scan operator is as follows.

Item	Description
TID Scan	Operator name.
TID Cond	Filter predicate of the operator. When a query is executed, rows that meet these conditions are included in the final result set.

6.2.3.2.6 Cte Scan

Description

Common table expression (CTE) is a temporary expression. CTE scan is used to scan temporary tables generated by CTE expressions.

In GaussDB, you can use the WITH keyword to specify one or more CTEs, which can be used multiple times in subsequent queries.

Typical Scenarios

When a query result set needs to be used for multiple times in subsequent queries, you can use CTE to avoid multiple calculations.

Examples

Example: CTEs with the WITH statement cannot be rewritten as subqueries.

```

-- Prepare data.
gaussdb=# CREATE TABLE employees(deptid integer, salary number);
    
```

```

CREATE TABLE
gaussdb=# CREATE TABLE managers(deptid integer);
CREATE TABLE
-- Execution result.
gaussdb=# EXPLAIN WITH table_b AS (SELECT deptid,sum(salary) dept_salary FROM employees GROUP BY
deptid)
SELECT m.deptid, b.dept_salary
FROM managers m,table_b b,table_b c
WHERE m.deptid = b.deptid AND m.deptid = c.deptid;
          QUERY PLAN
-----
Hash Join (cost=1.15..1.81 rows=20 width=36)
Hash Cond: (c.deptid = m.deptid)
CTE table_b
-> Data Node Scan on employees "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=20 width=36)
Node/s: All datanodes
-> CTE Scan on table_b c (cost=0.00..0.40 rows=20 width=4)
-> Hash (cost=0.90..0.90 rows=20 width=40)
-> Hash Join (cost=0.25..0.90 rows=20 width=40)
Hash Cond: (b.deptid = m.deptid)
-> CTE Scan on table_b b (cost=0.00..0.40 rows=20 width=36)
-> Hash (cost=0.00..0.00 rows=20 width=4)
-> Data Node Scan on managers "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=20
width=4)
Node/s: All datanodes
(13 rows)
-- Drop the table.
gaussdb=# DROP TABLE employees, managers;

```

6.2.3.2.7 Function Scan

Description

Function scan is used to execute a function and obtain tuples from the result set returned by the function.

Typical Scenarios

When a query needs to call a function and return a result set, the function scan operator can call the function and obtain tuples from the result set returned by the function.

Examples

Example: Call a system function.

```

gaussdb=# EXPLAIN SELECT * FROM generate_series(2,4);
          QUERY PLAN
-----
Function Scan on generate_series (cost=0.00..10.00 rows=1000 width=4)
(1 row)

```

6.2.3.2.8 Sample Scan

Description

Sample scan is a method of scanning tables. It can randomly extract some data from a table for query instead of scanning the entire table. Sample scan is used together with the TABLESAMPLE keyword in [TABLESAMPLE sampling_method (argument [, ...]) [REPEATABLE (seed)]] mode. This technology is usually used

in a query scenario in which a large table or random sampling is required, to improve query efficiency.

Typical Scenarios

When the `tablesample` syntax is used for query, **sampling_method** is set to **system** or **bernoulli**.

Examples

Example: Query the sampling data.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1 (c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100), 2, 3);
INSERT 0 100

-- Execution result.
gaussdb=# EXPLAIN SELECT c1 FROM t1 tablesample system(10);
QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.06..4.11 rows=2 width=32)
Node/s: All datanodes
-> Sample Scan on t1 (cost=0.00..4.02 rows=2 width=32)
    Sampling: system (10::real)
(4 rows)

-- Drop.
gaussdb=# DROP TABLE t1;
```

In the preceding example, the output of the sample scan operator is as follows.

Item	Description
Sample Scan	Operator name.
Sampling	Operator sampling mode. In the example, system is used as the sampling method, and the sampling ratio is 10%.

6.2.3.2.9 SubQuery Scan

Description

If the rule-based optimizer (RBO) does not optimize a statement that contains a subquery and is being executed, this operator executes the query plan tree of the subquery and then transfers the subquery result to the upper-layer query.

Typical Scenarios

If a statement contains a subquery, the SubQuery scan operator is generated to obtain tuples from the subquery.

Examples

```
-- Prepare data.
gaussdb=# CREATE TABLE t_distinct(a int, b int, c int, d int, e regproc);

-- Execution result.
gaussdb=# EXPLAIN (costs off, nodes off) SELECT COUNT(DISTINCT(a)), COUNT(DISTINCT(b)), d FROM
t_distinct GROUP BY c, d;

```

QUERY PLAN

```
-----
Streaming (type: GATHER)
-> Hash Join
  Hash Cond: ((public.t_distinct.c = subquery."?column?") AND (public.t_distinct.d = subquery.d))
-> GroupAggregate
  Group By Key: public.t_distinct.c, public.t_distinct.d
-> Sort
  Sort Key: public.t_distinct.c, public.t_distinct.d
-> Streaming(type: REDISTRIBUTE)
  -> Seq Scan on t_distinct

-> Hash
  -> Subquery Scan on subquery
  -> HashAggregate
  Group By Key: public.t_distinct.c, public.t_distinct.d
  -> Streaming(type: REDISTRIBUTE)
  -> HashAggregate
  Group By Key: public.t_distinct.c, public.t_distinct.d, public.t_distinct.a
  -> Seq Scan on t_distinct

(17 rows)

-- Execution result.
gaussdb=# EXPLAIN SELECT c1 FROM t1 tablesample system(10);

```

QUERY PLAN

```
-----
Streaming (type: GATHER) (cost=0.06..4.11 rows=2 width=32)
Node/s: All datanodes
-> Sample Scan on t1 (cost=0.00..4.02 rows=2 width=32)
  Sampling: system (10:real)

(4 rows)

-- Drop the table.
gaussdb=# DROP TABLE t_distinct;
```

6.2.3.2.10 Values Scan

Description

Reads the list of values specified in the **VALUES** clause and returns it as a set of virtual rows to the query planner. These virtual rows can be used by other scanners or operators, such as hash join or merge join.

Typical Scenarios

This operator provides a quick and simple way to specify a set of values without reading data from a table. This is useful during testing or debugging, or when you need to insert a small amount of data.

Examples

Example: The insert statement contains **VALUES**.

```
-- Prepare data.
gaussdb=# CREATE TABLE test_b (c1 number);
CREATE TABLE
```

```
-- Execution result.
gaussdb=# EXPLAIN INSERT INTO test_b(c1) VALUES ('1'),('2');
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.12..2.21 rows=1 width=32)
  Node/s: All datanodes
  -> Insert on test_b (cost=0.00..2.11 rows=4 width=32)
    -> Streaming(type: REDISTRIBUTE) (cost=0.00..0.11 rows=4 width=32)
        Spawn on: datanode2
        -> Values Scan on "*"VALUES*" (cost=0.00..0.03 rows=4 width=32)
(6 rows)

-- Drop the table.
gaussdb=# DROP TABLE test_b;
```

6.2.3.2.11 WorkTable Scan

Description

WorkTable scan is a memory-based query optimization technology. It can cache query results in the memory to improve query performance. When GaussDB executes a query statement, it stores the query result in a temporary table and uses WorkTable scan to scan the temporary table to obtain the query result. WorkTable scan reduces disk I/O operations and improves query performance. However, it also has some disadvantages, for example, a large amount of memory space is consumed, and a problem such as memory overflow may be caused. Therefore, you need to adjust and optimize the WorkTable scan according to actual situations.

Typical Scenarios

- Large table query: When a large table is queried, WorkTable scan can divide the query result into multiple work units. Each work unit processes a part of data, improving the query efficiency.
- Multi-table join query: When multiple tables need to be associated for a query, WorkTable scan can divide the data of each table into multiple work units. Each work unit processes a part of data, reducing the data volume of join operations and improving query efficiency.
- Partitioned table query: When a partitioned table is queried, WorkTable scan can divide the data of each partition into multiple work units. Each work unit processes a part of the data, improving the query efficiency.
- Statistics query of a large amount of data: When a large amount of data needs to be queried, WorkTable scan can divide the data into multiple work units. Each work unit processes a part of the data, improving the query efficiency.

Examples

Example: Recursive Union tables without indexes

```
-- Prepare data.
gaussdb=# CREATE TABLE t9(a int);
CREATE TABLE
gaussdb=# INSERT INTO t9 VALUES(1);
INSERT 0 1

-- Execution result.
gaussdb=# EXPLAIN WITH RECURSIVE tt AS (
```

```

SELECT a FROM t9
UNION ALL
SELECT a + 1 FROM tt WHERE a < 10)
SELECT * FROM tt;
      QUERY PLAN
-----
Streaming (type: GATHER) (cost=48.89..85.84 rows=721 width=4)
  Node/s: All datanodes
    -> CTE Scan on tt (cost=0.00..7.20 rows=721 width=4)
      -> Recursive Union (cost=0.00..44.89 rows=721 width=4)
        -> Seq Scan on t9 (cost=0.00..13.13 rows=20 width=4)
        -> WorkTable Scan on tt (cost=0.00..2.45 rows=70 width=4)
            Filter: (a < 10)
(7 rows)

-- Drop.
gaussdb=# DROP TABLE t9;

```

In the preceding example, the output of the WorkTable scan operator is as follows.

Item	Description
WorkTable Scan	Operator name.
Filter	Filter condition of the operator. In the example, a is less than 10.

6.2.3.3 Table Join Modes

6.2.3.3.1 Nested Loop Join

Description

A nested loop join is the simplest join method, which is implemented in all relational database systems. It compares the data in two tables to check whether the join conditions are met.

In GaussDB, a nested loop join scans the inner table for each row in the outer table searching for rows that meet the join conditions. This is similar to two nested loops. The outer loop traverses the outer table, and the inner loop traverses the inner table.

The time complexity of a nested loop join is $O(n \times m)$, where n and m indicate the numbers of rows in the two tables. If the inner table can be scanned by using indexes, the time complexity can be reduced to $O(n \log m)$.

Typical Scenarios

- Any of the inner or outer table is very small.
- An inner table quickly locates the row that meets the condition (for example, an index has been created for the join column of the inner table) based on the join condition.
- A nested loop join can be executed under any join conditions if it has no restrictions.

Examples

Example: Join two small tables.

```
-- Prepare data.
gaussdb=# CREATE TABLE employee(id int, deptid int);
CREATE TABLE
gaussdb=# INSERT INTO employee VALUES(1, 1), (2,1),(3,2),(4, 1), (5,2);
INSERT 0 5
gaussdb=# CREATE TABLE manager(id int, deptid int);
CREATE TABLE
gaussdb=# INSERT INTO manager VALUES(1,1), (2,2),(3,1),(4,2);
INSERT 0 4

-- Collect statistics.
gaussdb=# ANALYZE employee;
gaussdb=# ANALYZE manager;

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM employee e JOIN manager m ON e.deptid < m.deptid;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=4.00..38.43 rows=133 width=16)
Node/s: All datanodes
-> Nested Loop (cost=0.00..32.24 rows=133 width=16)
   Join Filter: (e.deptid < m.deptid)
   -> Streaming(type: BROADCAST) (cost=0.00..15.18 rows=40 width=8)
       Spawn on: All datanodes
       -> Seq Scan on employee e (cost=0.00..13.13 rows=20 width=8)
   -> Materialize (cost=0.00..13.20 rows=20 width=8)
       -> Seq Scan on manager m (cost=0.00..13.13 rows=20 width=8)
(9 rows)

-- Drop.
gaussdb=# DROP TABLE employee,manager;
```

In the preceding example, the output of nested loop join is as follows.

Item	Description
Nested Loop	Operator name.
Join Filter	Join predicate of the operator join. In the example, the condition is that e.deptid is less than m.deptid . During query execution, rows that meet these conditions are included in the final result set.

6.2.3.3.2 Hash Join

Description

A hash join is an efficient join method that depends on the hash technology. During a hash join, GaussDB selects one of the two tables (usually small tables) and creates a hash table based on join conditions. The key of the hash table is the join column of the small table, and the other columns of the small table are the value. Then, the hash value of the join column is calculated for each row in the large table. After this, the hash table is searched for a matching row.

The time complexity of a hash join is $O(n + m)$, where n and m indicate the numbers of rows in the two tables. However, the performance may degrade if the inner table is too large for the memory storage, as it will require additional disk I/O operations.

Typical Scenarios

- The number of rows in two tables differ significantly and the join operation is performed.
- The join columns of the two tables are equi-join (for example, using the = operator).

Examples

Example: The number of rows in two tables differ significantly and the join operation is performed.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
gaussdb=# CREATE TABLE t1(c1 number,c2 number,c3 number);
CREATE TABLE
gaussdb=# CREATE TABLE t2(c1 number,c2 number,c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100000), 2 ,3);
INSERT 0 100000
gaussdb=# INSERT INTO t2 VALUES(generate_series(1,100), 2, 3);
INSERT 0 100

-- Collect statistics.
gaussdb=# ANALYZE t1;
gaussdb=# ANALYZE t2;

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.c1 = t2.c2;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=12.56..986.88 rows=100 width=31)
  Node/s: All datanodes
  -> Hash Join (cost=9.44..982.19 rows=100 width=31)
      Hash Cond: (t1.c1 = t2.c2)
      -> Seq Scan on t1 (cost=0.00..816.00 rows=100000 width=16)
      -> Hash (cost=8.19..8.19 rows=100 width=15)
          -> Streaming(type: REDISTRIBUTE) (cost=0.00..8.19 rows=100 width=15)
              Spawn on: All datanodes
              -> Seq Scan on t2 (cost=0.00..2.50 rows=100 width=15)

(9 rows)

-- Drop the table.
gaussdb=# DROP TABLE t1,t2;
```

In the preceding example, the output of the hash join operator is as follows.

Item	Description
Hash Join	Operator name.

Item	Description
Hash Cond	Join predicate of the operator hash join. In the example, the condition is that t1.c1 is equal to t2.c2 . During query execution, rows that meet these conditions are included in the final result set.
Hash	Operator for creating a hash table in an inner table.

6.2.3.3.3 Merge Join

Description

A merge join is an efficient join method that depends on sorting operations. During a merge join, GaussDB sorts the join columns of the two tables and scans both tables at the same time to find matching rows. The time complexity of a merge join is $O(n + m)$, where n and m indicate the numbers of rows in the two tables. However, if a sorting operation is required, the time complexity of the sorting operation may reach $\max(O(\log n), O(\log m))$, which is usually more time-consuming than a direct merge join operation. In GaussDB, the optimizer prefers hash join, even if the two tables to be joined have been sorted.

Typical Scenarios

- The two tables are similar in size.
- The join columns of two tables have been sorted, for example, by index.
- In addition to the equi-join (for example, using the = operator), the join condition also includes the range query (for example, using the <, <=, >, and >= operators).

Examples

Example: Two tables are joined and the sizes of the two tables are close.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
gaussdb=# CREATE TABLE t1(id int,info text);
CREATE TABLE
gaussdb=# CREATE TABLE t2(id int,info text);
CREATE TABLE
gaussdb=# INSERT INTO t2 SELECT generate_series(1,100000),'bill'||generate_series(1,100000);
INSERT 0 100000
gaussdb=# INSERT INTO t1 SELECT generate_series(1,100000),'bill'||generate_series(1,100000);
INSERT 0 100000

-- Collect statistics.
gaussdb=# ANALYZE t1;
gaussdb=# ANALYZE t2;

-- Execution result.
gaussdb=# EXPLAIN SELECT /*+ MERGEJOIN(t2 t1) */ * FROM t2 JOIN t1 ON (t1.id=t2.id);
QUERY PLAN
```

```
-----
Streaming (type: GATHER) (cost=9352.82..15036.32 rows=100000 width=26)
Node/s: All datanodes
-> Merge Join (cost=9348.82..10348.82 rows=100000 width=26)
    Merge Cond: (t2.id = t1.id)
    -> Sort (cost=4674.41..4799.41 rows=100000 width=13)
        Sort Key: t2.id
        -> Seq Scan on t2 (cost=0.00..772.00 rows=100000 width=13)
    -> Sort (cost=4674.41..4799.41 rows=100000 width=13)
        Sort Key: t1.id
        -> Seq Scan on t1 (cost=0.00..772.00 rows=100000 width=13)
(10 rows)

-- Drop the table.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
```

In the preceding example, the output of the merge join operator is as follows.

Item	Description
Merge Join	Operator name.
Merge Cond	Join predicate of the operator merge join. In the example, the condition is that t2.id is equal to t1.id . During query execution, rows that meet these conditions are included in the final result set.

6.2.3.4 Operators

6.2.3.4.1 Sort

Description

Sorts the tuples returned by the bottom-layer node. The sort operator is used to sort query results based on specified sorting rules and return ordered result sets.

Typical Scenarios

- If a query statement contains the ORDER BY clause, GaussDB selects the sort operator in the execution plan to perform sorting.
- Merge join is used for join operations.

Examples

Example: The query statement contains the ORDER BY clause.

```
-- Prepare data.
gaussdb=# CREATE TABLE student(id integer, class_id integer, grade number);
CREATE TABLE
gaussdb=# INSERT INTO student VALUES(generate_series(1,50), 1, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(51,100), 2, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(101,150), 3, floor(100 * random()));
INSERT 0 50
```

```

gaussdb=# INSERT INTO student VALUES(generate_series(151,200), 3, floor(100 * random()));
INSERT 0 50

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM student ORDER BY grade;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=14.18..14.62 rows=21 width=40)
  Merge Sort Key: grade
  Node/s: All datanodes
  -> Sort (cost=13.37..13.40 rows=21 width=40)
      Sort Key: grade
      -> Seq Scan on student (cost=0.00..13.13 rows=20 width=40)
(6 rows)

-- Drop.
gaussdb=# DROP TABLE student;

```

In the preceding example, the output of the sort operator is as follows.

Item	Description
Sort	Operator name.
Sort Key	Keyword based on which the sort operator uses for sorting. In the example, the value is grade .

6.2.3.4.2 Limit

Description

The limit operator limits the number of output execution results. If a limit operator is added, not all rows are retrieved.

Typical Scenarios

Use a query statement containing the keyword limit.

Examples

Example: Use a query statement containing the keyword limit.

```

-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1 ( id int, number int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES (generate_series(1,50), 1);
INSERT 0 50
gaussdb=# EXPLAIN SELECT * FROM t1 LIMIT 10 OFFSET 20;
          QUERY PLAN
-----
Limit (cost=0.00..0.00 rows=1 width=8)
 -> Data Node Scan on "__REMOTE_LIMIT_QUERY__" (cost=0.00..0.00 rows=1 width=8)
    Node/s: All datanodes
(3 rows)

-- Drop.
gaussdb=# DROP TABLE t1;

```

6.2.3.4.3 Append

Description

Append is used to append multiple relationship sets and process the linked list that contains one or more subplans.

Typical Scenarios

- SQL statements with UNION.
- SQL statements with UNION ALL.

Examples

Example 1: SQL statement with UNION.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 100), 2, 3);
INSERT 0 100
gaussdb=# CREATE TABLE t2(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES(generate_series(1, 100), 2, 3);
INSERT 0 100

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1 UNION SELECT * FROM t2;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=28.03..28.94 rows=42 width=96)
Node/s: All datanodes
-> HashAggregate (cost=26.72..26.98 rows=42 width=96)
Group By Key: t1.c1, t1.c2, t1.c3
-> Append (cost=0.00..26.52 rows=42 width=96)
-> Seq Scan on t1 (cost=0.00..13.13 rows=20 width=96)
-> Seq Scan on t2 (cost=0.00..13.13 rows=20 width=96)
(7 rows)
```

Example 2: SQL statement with UNION ALL.

```
gaussdb=# EXPLAIN SELECT * FROM t1 UNION ALL SELECT * FROM t2;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=1.62..28.70 rows=40 width=96)
Node/s: All datanodes
-> Result (cost=0.00..26.26 rows=40 width=96)
-> Append (cost=0.00..26.26 rows=40 width=96)
-> Seq Scan on t1 (cost=0.00..13.13 rows=20 width=96)
-> Seq Scan on t2 (cost=0.00..13.13 rows=20 width=96)
(6 rows)

-- Drop the table.
gaussdb=# DROP TABLE t1,t2;
```

6.2.3.4.4 Agg

Description

The Agg operator is used to perform aggregation calculation. It supports three policies: common aggregation (aggregation without grouping), sort aggregation,

and hash aggregation. Sort aggregation and hash aggregation must be used together with GROUP BY because they involve grouping. The difference between sort aggregation and hash aggregation is that the input of sort aggregation must be ordered, while hash aggregation does not depend on the input sequence. Even if the input is sorted, sort aggregation may not be selected because hash aggregation may have better execution performance.

Typical Scenarios

- Common aggregation: Only aggregation calculation is involved. It is used to collect statistics on the total number of tables, or the number of rows or features in a column. The keyword Aggregate is used in the execution plan.
- Sort aggregation: If the input tuples are sorted or the group keys are ordered, the sort aggregation may be selected. The GroupAggregate keyword is used in the execution plan.
- Hash aggregation: It is the aggregation calculation in most service scenarios and in the case of disordered data. The HashAggregate keyword is used in the execution plan.

Examples

Example 1: common aggregation.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS student;
gaussdb=# CREATE TABLE student(id integer, class_id integer, grade number);
CREATE TABLE
gaussdb=# INSERT INTO student VALUES(generate_series(1,50), 1, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(51,100), 2, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(101,150), 3, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(151,200), 3, floor(100 * random()));
INSERT 0 50

-- Execution result.
gaussdb=# EXPLAIN SELECT count(*) FROM student;
          QUERY PLAN
-----
Aggregate  (cost=13.23..13.27 rows=1 width=8)
->  Streaming (type: GATHER) (cost=13.23..13.27 rows=2 width=8)
    Node/s: All datanodes
    -> Aggregate  (cost=13.16..13.17 rows=2 width=8)
        -> Seq Scan on student  (cost=0.00..13.13 rows=20 width=0)
(5 rows)
```

Example 2: sort aggregation.

```
gaussdb=# SET enable_hashagg = off;
SET
gaussdb=# EXPLAIN SELECT id, count(class_id) FROM student GROUP BY id ORDER BY id;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=14.00..14.54 rows=21 width=16)
Merge Sort Key: id
Node/s: All datanodes
-> GroupAggregate (cost=13.37..13.60 rows=21 width=16)
    Group By Key: id
    -> Sort (cost=13.37..13.40 rows=21 width=8)
        Sort Key: id
        -> Seq Scan on student  (cost=0.00..13.13 rows=20 width=8)
(8 rows)
```

Example 3: hash aggregation.

```
gaussdb=# SET enable_hashagg = on;
gaussdb=# SET enable_enable_fast_query_shipping = on;
gaussdb=# EXPLAIN SELECT id, avg(grade) FROM student GROUP BY id;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=14.00..14.57 rows=21 width=68)
Node/s: All datanodes
-> GroupAggregate (cost=13.37..13.63 rows=21 width=68)
   Group By Key: id
   -> Sort (cost=13.37..13.40 rows=21 width=36)
       Sort Key: id
       -> Seq Scan on student (cost=0.00..13.13 rows=20 width=36)
(7 rows)

-- Drop.
gaussdb=# DROP TABLE student;
```

In the preceding example, the output of the aggregate operator is as follows.

Item	Description
Aggregate	Name of a common aggregation operator, indicating that only aggregation is performed without grouping.
GroupAggregate	Name of a sort aggregation operator, indicating that the input tuple is sorted or the grouping key is ordered.
HashAggregate	Name of a hash aggregation operator, which is used for aggregation calculation in most scenarios and in the case of disordered data.

6.2.3.4.5 Group

Description

The Group operator is used to process the GROUP BY clause and group the lower-layer sorted tuples. The returned result is the result after grouping by group key.

Typical Scenarios

Grouping: queries the number of different values in a column. The function of this operation is similar to that of DISTINCT.

Examples

Example: The query statement contains the GROUP BY clause.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS student;
gaussdb=# CREATE TABLE student(id integer, class_id integer, grade number);
CREATE TABLE
gaussdb=# INSERT INTO student VALUES(generate_series(1,50), 1, floor(100 * random()));
```

```

INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(51,100), 2, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(101,150), 3, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(151,200), 3, floor(100 * random()));
INSERT 0 50

-- Execution result.
gaussdb=# SET enable_hashagg = off;
SET
gaussdb=# EXPLAIN SELECT class_id FROM student GROUP BY class_id ORDER BY class_id;
          QUERY PLAN
-----
 Group  (cost=13.37..13.44 rows=26 width=4)
  Group By Key: class_id
    -> Sort  (cost=15.27..15.33 rows=26 width=4)
        Sort Key: class_id
        -> Streaming (type: GATHER) (cost=14.18..14.65 rows=26 width=4)
            Node/s: All datanodes
            -> Group  (cost=13.37..13.44 rows=26 width=4)
                Group By Key: class_id
                -> Sort  (cost=13.37..13.40 rows=21 width=4)
                    Sort Key: class_id
                    -> Seq Scan on student  (cost=0.00..13.13 rows=20 width=4)

(11 rows)

-- Drop.
gaussdb=# DROP TABLE student;

```

In the preceding example, the output of the Group operator is as follows.

Item	Description
Group	Operator name.
Group By Key	Group keyword. In the example, class_id is used as the keyword.

6.2.3.4.6 MergeAppend

Description

MergeAppend is used to append multiple ordered relationship sets. The operation is similar to Append. The only difference is that MergeAppend accelerates the calculation of ordered relationship sets. MergeAppend merges subquery results in a way that preserves the sort order. It can be used to merge sorted rows in a table partition. Therefore, different from common Append, MergeAppend needs to ensure that the input **m_plan** is ordered before performing operations. Generally, a sort operator appears before the MergeAppend operation (that is, in the subtree of the execution plan).

Typical Scenarios

The partition scan path is index or index-only, the partition pruning result is greater than 1, and the following conditions are met:

All partition indexes are valid B-tree indexes. The SQL query contains the LIMIT clause. During partition scan, no partitioned table query statement with the filter exists.

The MergeAppend path is no longer generated when the GUC parameter `sql_beta_feature` is set to `'disable_merge_append_partition'`.

Examples

```
-- Prepare data.
gaussdb=# CREATE TABLE test_range_pt (a INT, b INT, c INT)
PARTITION BY RANGE(a)
(
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN (3000),
  PARTITION p3 VALUES LESS THAN (4000),
  PARTITION p4 VALUES LESS THAN (5000),
  PARTITION p5 VALUES LESS THAN (MAXVALUE)
)ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO test_range_pt VALUES
(generate_series(1,10000),generate_series(1,10000),generate_series(1,10000));
INSERT 0 10000
gaussdb=# CREATE INDEX idx_range_b ON test_range_pt(b) LOCAL;
CREATE INDEX
gaussdb=# ANALYZE test_range_pt;
ANALYZE

-- Execution result.
gaussdb=# EXPLAIN ANALYZE SELECT * FROM test_range_pt WHERE b >10 AND b < 5000 ORDER BY b
LIMIT 10;

                                QUERY PLAN
-----
Limit  (cost=153.32..153.49 rows=10 width=12) (actual time=9.131..9.138 rows=10 loops=1)
-> Streaming (type: GATHER) (cost=153.32..153.66 rows=10 width=12) (actual time=9.127..9.130
rows=10 loops=1)
    Merge Sort Key: b
    Node/s: All datanodes
    -> Limit  (cost=152.70..152.72 rows=20 width=12) (actual time=[4.777,4.782]..[4.899,4.903], rows=20)
        -> Sort  (cost=152.70..158.94 rows=20 width=12) (actual time=[4.774,4.776]..[4.897,4.899],
rows=20)
            Sort Key: b
            Sort Method: top-N heapsort  Memory: 26kB ~ 26kB
            -> Partition Iterator  (cost=0.00..98.76 rows=4991 width=12) (actual time=[0.164,3.639]..
[0.179,3.743], rows=4989)
                Iterations: 5
                -> Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..98.76
rows=4991 width=12) (actual time=[0.522,2.923]..[0.602,2.980], rows=4989)
                    Index Cond: ((b > 10) AND (b < 5000))
                    Selected Partitions: 1..5
Total runtime: 10.534 ms
(14 rows)

-- Drop.
gaussdb=# DROP TABLE test_range_pt;
```

6.2.3.4.7 SetOp

Description

A SetOp operator is used to combine two or more query results into a result set. SetOp operators include INTERSECT and EXCEPT.

Typical Scenarios

- **INTERSECT**: returns the intersection of two query results, that is, the rows that exist in both result sets.
- **INTERSECT ALL**: returns the intersection of two query results, including duplicate rows.
- **EXCEPT**: returns the rows that exist in the first query result but do not exist in the second query result.
- **EXCEPT ALL**: returns the rows that exist in the first query result but do not exist in the second query result, including duplicate rows.

Examples

Example 1: INTERSECT.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t;
gaussdb=# CREATE TABLE t(a int, b int, c int);
CREATE TABLE
gaussdb=# INSERT INTO t VALUES(generate_series(1, 10), generate_series(601, 610), generate_series(901, 910));
INSERT 0 10

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t INTERSECT SELECT * FROM t;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=27.94..28.61 rows=21 width=12)
Node/s: All datanodes
-> SetOp Intersect (cost=27.13..27.39 rows=21 width=12)
-> Sort (cost=27.13..27.20 rows=42 width=12)
    Sort Key: *(SELECT* 1".a, *(SELECT* 1".b, *(SELECT* 1".c
-> Append (cost=0.00..26.52 rows=42 width=12)
    -> Subquery Scan on *(SELECT* 1" (cost=0.00..13.26 rows=21 width=12)
        -> Seq Scan on t (cost=0.00..13.13 rows=20 width=12)
    -> Subquery Scan on *(SELECT* 2" (cost=0.00..13.26 rows=21 width=12)
        -> Seq Scan on t (cost=0.00..13.13 rows=20 width=12)

(10 rows)
```

In the preceding example, the output of the SetOp operator is as follows.

Item	Description
SetOp	Operator name.
Intersect	Mode of merging result sets. In the example, Intersect indicates the intersection of two query results, that is, the rows that exist in both result sets.

Example 2: INTERSECT ALL.

```
gaussdb=# EXPLAIN SELECT * FROM t INTERSECT ALL SELECT * FROM t;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=27.94..28.61 rows=21 width=12)
Node/s: All datanodes
-> SetOp Intersect All (cost=27.13..27.39 rows=21 width=12)
```

```

-> Sort (cost=27.13..27.20 rows=42 width=12)
    Sort Key: ""SELECT* 1".a, ""SELECT* 1".b, ""SELECT* 1".c
-> Append (cost=0.00..26.52 rows=42 width=12)
    -> Subquery Scan on ""SELECT* 1" (cost=0.00..13.26 rows=21 width=12)
        -> Seq Scan on t (cost=0.00..13.13 rows=20 width=12)
    -> Subquery Scan on ""SELECT* 2" (cost=0.00..13.26 rows=21 width=12)
        -> Seq Scan on t (cost=0.00..13.13 rows=20 width=12)
(10 rows)

```

In the preceding example, the output of the SetOp operator is as follows.

SetOp	Operator name.
Intersect All	Mode of merging result sets. In the example, Intersect All indicates that the intersection of two query results is returned, including duplicate rows.

Example 3: EXCEPT.

```

gaussdb=# EXPLAIN SELECT * FROM t EXCEPT SELECT * FROM t;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=27.94..28.61 rows=21 width=12)
 Node/s: All datanodes
-> SetOp Except (cost=27.13..27.39 rows=21 width=12)
   -> Sort (cost=27.13..27.20 rows=42 width=12)
       Sort Key: ""SELECT* 1".a, ""SELECT* 1".b, ""SELECT* 1".c
   -> Append (cost=0.00..26.52 rows=42 width=12)
       -> Subquery Scan on ""SELECT* 1" (cost=0.00..13.26 rows=21 width=12)
           -> Seq Scan on t (cost=0.00..13.13 rows=20 width=12)
       -> Subquery Scan on ""SELECT* 2" (cost=0.00..13.26 rows=21 width=12)
           -> Seq Scan on t (cost=0.00..13.13 rows=20 width=12)
(10 rows)

```

In the preceding example, the output of the SetOp operator is as follows.

Item	Description
SetOp	Operator name.
Except	Mode of merging result sets. In the example, Except indicates that the rows that exist in the first query result but do not exist in the second query result are returned.

Example 4: EXCEPT ALL.

```

gaussdb=# EXPLAIN SELECT * FROM t EXCEPT ALL SELECT * FROM t;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=27.94..28.61 rows=21 width=12)
 Node/s: All datanodes
-> SetOp Except All (cost=27.13..27.39 rows=21 width=12)
   -> Sort (cost=27.13..27.20 rows=42 width=12)
       Sort Key: ""SELECT* 1".a, ""SELECT* 1".b, ""SELECT* 1".c
   -> Append (cost=0.00..26.52 rows=42 width=12)

```

```

-> Subquery Scan on ""SELECT* 1" (cost=0.00..13.26 rows=21 width=12)
   -> Seq Scan on t (cost=0.00..13.13 rows=20 width=12)
-> Subquery Scan on ""SELECT* 2" (cost=0.00..13.26 rows=21 width=12)
   -> Seq Scan on t (cost=0.00..13.13 rows=20 width=12)
(10 rows)

-- Drop.
gaussdb=# DROP TABLE t;

```

In the preceding example, the output of the SetOp operator is as follows.

Item	Description
SetOp	Operator name.
Except All	Mode of merging result sets. In the example, Except All indicates the rows that exist in the first query result but do not exist in the second query result, including duplicate rows.

6.2.3.4.8 RecursiveUnion

Description

The RecursiveUnion operator is used to process UNION statements that are recursively called. Such statements are usually used in CTEs. Common syntax logic is as follows: An initial input set is used as the initial data of the recursive process, and then recursive calling is performed to obtain the output. Finally, the output of the current recursive calling is used as the input of the next recursive calling, and the final output is obtained through cyclic calling.

Typical Scenarios

Execute an SQL statement with RecursiveUnion.

Examples

Example: Execute an SQL statement with RecursiveUnion.

```

-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 100), 2, 3);
INSERT 0 100

-- Execution result.
gaussdb=# EXPLAIN WITH RECURSIVE t1(n) AS (
VALUES(1)
UNION ALL
SELECT n+1 FROM t1 WHERE n < 100)
SELECT sum(n) FROM t1;
          QUERY PLAN
-----
Aggregate (cost=3.65..3.66 rows=1 width=12)
  CTE t1
    -> Recursive Union (cost=0.00..2.96 rows=31 width=4)
      -> Values Scan on ""VALUES"" (cost=0.00..0.01 rows=1 width=4)

```

```
-> WorkTable Scan on t1 (cost=0.00..0.23 rows=3 width=4)
    Filter: (n < 100)
-> CTE Scan on t1 (cost=0.00..0.62 rows=31 width=4)
(7 rows)
-- Drop.
gaussdb=# DROP TABLE IF EXISTS t1;
```

6.2.3.4.9 Unique

Description

Deduplicates lower-layer data. During the execution, it traverses all input data, filters duplicate records, and retains only the unique record.

Typical Scenarios

Disable the **enable_hashagg** parameter and perform the query with DISTINCT.

Examples

Example: Perform the query with DISTINCT.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1 (id INT , number INT);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,50), 1);
INSERT 0 50
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,50), 2);
INSERT 0 50

-- Execution result.
gaussdb=# SET enable_hashagg = off;
SET
gaussdb=# EXPLAIN SELECT DISTINCT t1.id FROM t1;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=14.18..14.65 rows=26 width=4)
  Node/s: All datanodes
  -> Unique (cost=13.37..13.44 rows=26 width=4)
    -> Sort (cost=13.37..13.40 rows=21 width=4)
        Sort Key: id
        -> Seq Scan on t1 (cost=0.00..13.13 rows=20 width=4)
(6 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS t1;
```

6.2.3.4.10 LockRows

Description

The LockRows operator is used to lock rows in a query result set to prevent other transactions from modifying or deleting these rows.

Typical Scenarios

- When SELECT ... FOR SHARE/UPDATE is used to lock rows in a transaction, other transactions cannot modify or delete these rows.

- When FOR SHARE is used to lock a row, the current transaction and other transactions cannot modify or delete the locked row.
- When FOR UPDATE is used to lock a row, no transaction can modify or delete the locked row except the current transaction.

Examples

Example: SELECT FOR UPDATE syntax.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t;
gaussdb=# CREATE TABLE t(a int, b int, c int);
CREATE TABLE
gaussdb=# INSERT INTO t VALUES(generate_series(1, 10), generate_series(601, 610), generate_series(901, 910));
INSERT 0 10

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t WHERE a = 1 FOR UPDATE;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.06..13.26 rows=1 width=18)
Node/s: All datanodes
-> LockRows (cost=0.00..13.16 rows=1 width=18)
   -> Streaming(type: REDISTRIBUTE) (cost=0.00..13.16 rows=1 width=18)
       Spawn on: datanode1
       -> Seq Scan on t (cost=0.00..13.16 rows=1 width=18)
           Filter: (a = 1)
(7 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS t;
```

6.2.3.4.11 Materialize

Description

The Materialize operator is used to cache the results returned by subnodes and save the subquery results. For subnodes that need to be scanned repeatedly for multiple times (especially when the scanning results are the same each time), the execution cost can be reduced.

Typical Scenarios

- If a query statement involves subqueries and the same batch of data needs to be queried for multiple times, the optimizer selects the Materialize operator to cache the subquery results, greatly reducing the scan execution time. Subqueries often exist in clauses such as IN, ANY, ALL, and EXISTS.
- Nested loop is used as the join operator.

Examples

Example: Perform subqueries with ALL.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS student;
gaussdb=# CREATE TABLE student(id integer, class_id integer, grade number);
CREATE TABLE
gaussdb=# CREATE TABLE class_table(class_id integer);
CREATE TABLE
```

```
gaussdb=# INSERT INTO student VALUES(generate_series(1,50), 1, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(51,100), 2, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(101,150), 3, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(151,200), 3, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO class_table VALUES(1),(2),(3),(4);
INSERT 0 4

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM student WHERE class_id >= all (SELECT class_id FROM class_table);
QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.38..100.67 rows=10 width=40)
Node/s: All datanodes
-> Seq Scan on student (cost=0.00..100.11 rows=10 width=40)
    Filter: (SubPlan 1)
        SubPlan 1
            -> Materialize (cost=0.00..13.28 rows=80 width=4)
                -> Streaming(type: BROADCAST) (cost=0.00..13.18 rows=40 width=4)
                    Spawn on: All datanodes
                        -> Seq Scan on class_table (cost=0.00..13.13 rows=20 width=4)
(9 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS student,class_table;
```

6.2.3.4.12 Result

Description

Result is used to process the condition expression ($2 > 1$) that needs to be calculated only once or the insert statement that contains only one VALUES clause. In this case, the process can be returned in advance and no subsequent operation is required.

Typical Scenarios

- The Result node is used to optimize the query of constant condition expressions. The condition expression does not depend on the scanned data, for example, **select * from t1 where 1 > 2**.
- If the SELECT statement does not contain the FROM clause or the INSERT statement contains only one VALUES clause, the executor directly calculates the projection attribute of SELECT or uses the VALUES clause to construct a tuple.

Examples

Example 1: The Result node is used to optimize the query of constant condition expressions.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 100), 2, 3);
INSERT 0 100

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE 1 > 2;
```

```

QUERY PLAN
-----
Result (cost=0.00..0.01 rows=1 width=0)
  One-Time Filter: false
(2 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS t1;
    
```

In the preceding example, the output of the Result operator is as follows.

Item	Description
Result	Operator name.
One-Time Filter	The constant expression is always false during filtering. Therefore, the expression needs to be executed only once.

Example 2: SELECT does not contain the FROM clause.

```

gaussdb=# EXPLAIN SELECT dbe_raw.bit_and('1','2');
QUERY PLAN
-----
Result (cost=0.00..0.01 rows=1 width=0)
(1 row)
    
```

Example 3: The INSERT statement contains only one VALUES clause.

```

gaussdb=# EXPLAIN INSERT INTO t1 VALUES(1, 2, 3);
QUERY PLAN
-----
Insert on t1 (cost=0.00..0.01 rows=1 width=0)
  Node/s: All datanodes
  Node expr: c1
  -> Result (cost=0.00..0.01 rows=1 width=0)
(4 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS t1;
    
```

6.2.3.4.13 WindowAgg

Description

The WindowAgg operator is used to process tuple window aggregation. The functions and implementation modes of the WindowAgg operator are similar to those of the aggregate operator. The main difference is that the tuples processed by the WindowAgg operator are restricted to the same window, while the tuples processed by the aggregate operator are the entire table (GROUP BY division).

Typical Scenarios

The query statement contains window functions, such as row_number() OVER (PARTITION BY xxx) and avg(xxx) OVER (PARTITION BY xxx).

Examples

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t;
gaussdb=# CREATE TABLE t(a int, b int, c int);
CREATE TABLE
gaussdb=# INSERT INTO t VALUES(generate_series(1, 10), generate_series(601, 610), generate_series(901, 910));
INSERT 0 10

-- Execution result.
gaussdb=# EXPLAIN SELECT a, avg(b) OVER(partition by a) FROM t;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=14.18..14.82 rows=21 width=8)
 Node/s: All datanodes
  -> WindowAgg (cost=13.37..13.60 rows=21 width=8)
     -> Sort (cost=13.37..13.40 rows=21 width=8)
         Sort Key: a
         -> Seq Scan on t (cost=0.00..13.13 rows=20 width=8)
(6 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS t;
```

6.2.3.4.14 Rownum

Description

The Rownum operator generates a pseudocolumn that returns a number indicating the row number of the result obtained from the query. The value of **Rownum** in the first row is **1**. You can use the keyword Rownum to filter the row number in the query result. It usually appears in the ROWNUM clause.

The restrictions on using ROWNUM are as follows:

- Do not use ROWNUM as an alias to avoid ambiguity in SQL statements.
- Do not use ROWNUM when creating an index.
- Do not use ROWNUM as the default value when creating a table.
- Do not use ROWNUM as an alias in the WHERE clause.
- Do not use ROWNUM when inserting data.
- Do not use ROWNUM in a tableless query.
- Do not use ROWNUM in the LIMIT clause.
- Do not use ROWNUM as a parameter of the EXECUTE statement.
- Do not use ROWNUM to update a clause in the UPSERT statement.

Typical Scenarios

The filter condition statement contains ROWNUM.

Examples

Example: The filter condition statement contains ROWNUM.

```
-- Prepare data.
gaussdb=# CREATE TABLE t1 ( id INT , number INT );
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,50), 1);
```

```

INSERT 0 50
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,50), 2);
INSERT 0 50

-- Execution result.
gaussdb=# EXPLAIN SELECT t1.id FROM t1 WHERE rownum > 10;
          QUERY PLAN
-----
 Rownum  (cost=0.00..2.50 rows=150 width=4)
  Filter: (ROWNUM > 10)
    -> Seq Scan on t1  (cost=0.00..2.50 rows=150 width=4)
    (3 rows)

-- Drop.
gaussdb=# DROP TABLE t1;
    
```

In the preceding example, the output of the Rownum operator is as follows.

Item	Description
Rownum	Operator name.
Filter	ROWNUM predicate for filtering. In the example, ROWNUM > 10 indicates that the filter condition is to display rows whose ROWNUM is greater than 10.

6.2.3.4.15 Unpivot

Description

Transpose operator, which is used to convert rows into columns. This operation can convert data in multiple columns in a table into two columns. One column is the column name in the original table, and the other column is the corresponding value.

Typical Scenarios

Query the table that uses unpivot transposition.

Examples

```

Example: Query the table that uses unpivot transposition.
-- The UNPIVOT syntax can be used only in the O-compatible database.
gaussdb=# CREATE DATABASE ora_test WITH DBCOMPATIBILITY = 'ORA';
gaussdb=# \c ora_test
gaussdb=# SET enable_fast_query_shipping=off;

-- Prepare data.
ora_test=# DROP TABLE IF EXISTS t1;
ora_test=# CREATE TABLE t1 (id int, number int, grade int);
CREATE TABLE
ora_test=# INSERT INTO t1 VALUES(generate_series(1,100), 1, 2);
INSERT 0 100

-- Execution result.
ora_test=# EXPLAIN SELECT * FROM t1 UNPIVOT (v1 FOR v2 in (id,number,grade));
          oracle
    
```

```

QUERY PLAN
-----
Streaming (type: GATHER) (cost=1.88..16.24 rows=60 width=36)
  Node/s: All datanodes
  -> Subquery Scan on __unnamed_unpivot_subquery__ (cost=0.00..13.43 rows=60 width=36)
    Filter: (__unnamed_unpivot_subquery__.v1 IS NOT NULL)
    -> Unpivot (cost=0.00..13.13 rows=60 width=36)
      -> Seq Scan on t1 (cost=0.00..13.13 rows=20 width=12)
(6 rows)

-- Drop.
ora_test=# DROP TABLE t1;
ora_test=# \c gaussdb
gaussdb=# DROP DATABASE ora_test;

```

6.2.3.5 Distributed Operators

6.2.3.5.1 Data Node Scan

Description

If the GUC parameter **enable_fast_query_shipping** is set to **on**, the optimizer attempts to push down the query statement from the CN to DNs for execution, and then returns the execution result to the DNs.

Typical Scenarios

Use the statements that can be pushed down.

Examples

Queries that can be pushed down

```

-- Prepare data.
gaussdb=# SET enable_fast_query_shipping=on;
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1 (id int, number int, grade int);

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1 where id > 1;
QUERY PLAN
-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
  Node/s: All datanodes
(2 rows)

```

The actual plan is generated on the DN. To view the execution plan on the DN, set the **max_datanode_for_plan** parameter.

```

-- Enable a parameter.
gaussdb=# SET max_datanode_for_plan = 1;
-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1 where id > 1;
QUERY PLAN
-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
  Node/s: All datanodes

Remote SQL: SELECT id, "number", grade FROM public.t1 WHERE id > 1
Datanode Name: datanode1
  Seq Scan on t1 (cost=0.00..34.31 rows=648 width=12)
    Filter: (id > 1)
(8 rows)

```

```
-- Drop.
gaussdb=# DROP TABLE IF EXISTS t1;
```

Item	Description
Data Node Scan	Specifies the operator name.
Node/s: All datanodes	Delivers data to all DNs.

6.2.3.5.2 Streaming

Description

If an SQL statement cannot be pushed down, a distributed execution plan is generated first. Streaming has three forms, which correspond to different data shuffle functions in the distributed structure.

- Streaming (type: GATHER): The CN collects data from DNs.
- Streaming (type: REDISTRIBUTE): Data is redistributed to all the DNs based on selected columns.
- Streaming (type: BROADCAST): Data on the current DN is broadcast to other DNs.

Typical Scenarios

Distributed Execution Plan

Examples

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
gaussdb=# CREATE TABLE t1(c1 int, c2 int);
gaussdb=# CREATE TABLE t2(c1 int, c2 int);

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1,t2 WHERE t1.c2 = t2.c2;
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=13.92..29.57 rows=20 width=16)
Node/s: All datanodes
-> Hash Join (cost=13.29..28.64 rows=20 width=16)
   Hash Cond: (t1.c2 = t2.c2)
   -> Streaming(type: BROADCAST) (cost=0.00..15.18 rows=40 width=8)
       Spawn on: All datanodes
       -> Seq Scan on t1 (cost=0.00..13.13 rows=20 width=8)
   -> Hash (cost=13.13..13.13 rows=21 width=8)
       -> Seq Scan on t2 (cost=0.00..13.13 rows=20 width=8)
(9 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
```

Item	Description
Streaming	Specifies the operator name.

Item	Description
Spawn on: All datanodes	Distributes data to all DNs.

6.2.3.5.3 Remote query

Description

If an SQL statement cannot be pushed down and a distributed execution plan cannot be generated, an execution plan for sending statements is generated. Run the **EXPLAIN VERBOSE** command to view the SQL statements delivered to DNs.

Typical Scenarios

If an SQL statement cannot be pushed down, a distributed execution plan cannot be generated.

Examples

```
-- Prepare the environment.
gaussdb=# CREATE TABLE t1(c1 int,c2 int);

gaussdb=# EXPLAIN VERBOSE SELECT * FROM t1 limit 1;
          QUERY PLAN
-----
Limit  (cost=0.00..0.00 rows=1 width=8)
Output: t1.c1, t1.c2
-> Data Node Scan on "__REMOTE_LIMIT_QUERY__" (cost=0.00..0.00 rows=1 width=8)
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM ONLY public.t1 WHERE true LIMIT 1::bigint
(6 rows)

-- Drop.
gaussdb=# DROP TABLE t1;
```

6.2.3.6 Partition Pruning

6.2.3.6.1 Partition Iterator

Description

Table partitioning logically divides a large table or index into smaller and easier-to-manage logical units (partitions), minimizing the impact on table query and modification statements. Users can quickly locate a partition where data is located by using a partition key. In this way, users do not need to scan all large tables in the database and can concurrently perform DDL and DML operations on different partitions. GaussDB supports three partitioning policies: range partitioning, hash partitioning, and list partitioning. Range partitioning is implemented based on binary-search, and the complexity is $O(\log N)$. Hash partitioning and list partitioning are implemented based on the key-partOid hash table, and the complexity is $O(1)$.

Typical Scenarios

- Create a level-2 partitioned table using range partitioning.
- Create a level-2 partitioned table using hash partitioning.
- Create a level-2 partitioned table using list partitioning.

Examples

Example 1: Create a level-2 partitioned table using range partitioning.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1 (c1 int, c2 int) PARTITION BY RANGE (c1) SUBPARTITION BY RANGE (c2)
(
  PARTITION p1 VALUES LESS THAN(10)
  (
    SUBPARTITION subp1_1 VALUES LESS THAN(5),
    SUBPARTITION subp1_2 VALUES LESS THAN(10)
  ),
  PARTITION p2 VALUES LESS THAN(20)
  (
    SUBPARTITION subp2_1 VALUES LESS THAN(15),
    SUBPARTITION subp2_2 VALUES LESS THAN(20)
  ),
  PARTITION p3 VALUES LESS THAN(MAXVALUE)
  (
    SUBPARTITION subp3_1 VALUES LESS THAN(30),
    SUBPARTITION subp3_2 VALUES LESS THAN(MAXVALUE)
  )
);
CREATE TABLE

gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100,2), 1);
INSERT 0 50
gaussdb=# INSERT INTO t1 VALUES(generate_series(2,101,2), 2);
INSERT 0 50

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 < 15 AND c2 <= 1;
QUERY PLAN
-----
Partition Iterator (cost=0.00..33.23 rows=239 width=8)
  Iterations: 2, Sub Iterations: 3
  -> Partitioned Seq Scan on t1 (cost=0.00..33.23 rows=239 width=8)
    Filter: ((c1 < 15) AND (c2 <= 1))
    Selected Partitions: 1..2
    Selected Subpartitions: 1:1..2 2:1
(6 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS t1;
```

In the preceding example, the output of the Partition Iterator operator is as follows.

Item	Description
Partition Iterator	Operator name.

Item	Description
Iterations	Specifies the number of iterations performed by the partition iteration operator on level-1 partitions. If PART is displayed, dynamic pruning is used. For example, Iterations: 2 in example 1 indicates that the iteration operator needs to traverse two level-1 partitions. Iterations: PART indicates that the number of level-1 partitions to be traversed is determined by parameter conditions of the partition key.
Sub Iterations	Specifies the number of iterations performed by the partition iteration operator on level-2 partitions. If PART is displayed, dynamic pruning is used. For example, Sub Iterations: 3 in example 1 indicates that the iteration operator needs to traverse three level-2 partitions. Iterations: PART indicates that the number of level-2 partitions to be traversed is determined by parameter conditions of the partition key.
Filter	Filter predicate of the operator. In the example, the filter condition is that the value in column c1 is less than 15 and the value in column c2 is less than or equal to 1. When a query is executed, rows that meet these conditions are included in the final result set.
Partitioned Seq Scan	Scan mode of a partitioned table.
Selected Partitions	Level-1 partition pruning result. In example 1, Selected Partitions: 1..2 indicates that partitions 1 and 2 are selected.
Selected Subpartitions	Level-2 partition pruning result. In example 1, Selected Subpartitions: 1:1..2 2:1 indicates that subpartitions 1 and 2 of the first level-1 partition and partition 1 of the second level-1 partition are selected. Selected Subpartitions: ALL indicates that all level-2 partitions are selected.

Example 2: Create a level-2 partitioned table using hash partitioning.

```

-- Prepare data.
gaussdb=# CREATE TABLE sales (id INT, number INT)
PARTITION BY HASH (number)
SUBPARTITION BY HASH (id)
(
  PARTITION p0
  (
    SUBPARTITION sp0_0,
    SUBPARTITION sp0_1
  ),
  PARTITION p1
  (
    SUBPARTITION sp1_0,
    SUBPARTITION sp1_1
  )
);
CREATE TABLE

gaussdb=# INSERT INTO sales VALUES(generate_series(1,50), generate_series(1,50));
INSERT 0 50

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM sales WHERE id < 15 AND number <= 50;
          QUERY PLAN
-----
Partition Iterator  (cost=0.00..42.23 rows=239 width=8)
  Iterations: 2, Sub Iterations: 4
  -> Partitioned Seq Scan on sales  (cost=0.00..42.23 rows=239 width=8)
       Filter: ((id < 15) AND ("number" <= 50))
       Selected Partitions:  1..2
       Selected Subpartitions:  ALL
(6 rows)

-- Drop.
DROP TABLE sales;

```

In the preceding example, the output of the Partition Iterator operator is as follows.

Item	Description
Partition Iterator	Operator name.
Iterations	Specifies the number of iterations performed by the partition iteration operator on level-1 partitions. If PART is displayed, dynamic pruning is used. For example, Iterations: 2 in example 2 indicates that the iteration operator needs to traverse two level-1 partitions. Iterations: PART indicates that the number of level-1 partitions to be traversed is determined by parameter conditions of the partition key.

Item	Description
Sub Iterations	Specifies the number of iterations performed by the partition iteration operator on level-2 partitions. If PART is displayed, dynamic pruning is used. For example, Sub Iterations: 4 in example 1 indicates that the iteration operator needs to traverse four level-2 partitions. Iterations: PART indicates that the number of level-2 partitions to be traversed is determined by parameter conditions of the partition key.
Filter	Filter predicate of the operator. In the example 2, the filter condition is that the value in column id is less than 15 and the value in column number is less than or equal to 50. When a query is executed, rows that meet these conditions are included in the final result set.
Partitioned Seq Scan	Scan mode of a partitioned table.
Selected Partitions	Level-1 partition pruning result. In example 2, Selected Partitions: 1..2 indicates that partitions 1 and 2 are selected.
Selected Subpartitions	Level-2 partition pruning result. Selected Subpartitions: ALL in example 2 indicates that all level-2 partitions are selected.

Example 3: Create a level-2 partitioned table using list partitioning.

```
-- Prepare data.
gaussdb=# CREATE TABLE list_partitioned_table (id INT, category INT)
PARTITION BY LIST (category)
SUBPARTITION BY LIST (id)
(
  PARTITION p0 VALUES (1, 3, 5, 7)
  (
    SUBPARTITION sp0_0 VALUES (0),
    SUBPARTITION sp0_1 VALUES (1)
  ),
  PARTITION p1 VALUES (2, 4, 6, 8)
  (
    SUBPARTITION sp1_0 VALUES (0),
    SUBPARTITION sp1_1 VALUES (1)
  )
);
CREATE TABLE
gaussdb=# INSERT INTO list_partitioned_table VALUES(0,generate_series(1,8,2));
```

```

INSERT 0 4
gaussdb=# INSERT INTO list_partitioned_table VALUES(1,generate_series(2,7,2));
INSERT 0 3

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM list_partitioned_table WHERE id = 0 AND category <= 5;
          QUERY PLAN
-----
Partition Iterator  (cost=0.00..42.23 rows=4 width=8)
  Iterations: 2, Sub Iterations: 2
  -> Partitioned Seq Scan on list_partitioned_table  (cost=0.00..42.23 rows=4 width=8)
       Filter: ((category <= 5) AND (id = 0))
       Selected Partitions:  1..2
       Selected Subpartitions: 1:1 2:1
(6 rows)

-- Drop.
gaussdb=# DROP TABLE list_partitioned_table;
    
```

In the preceding example, the output of the Partition Iterator operator is as follows.

Item	Description
Partition Iterator	Operator name.
Iterations	Specifies the number of iterations performed by the partition iteration operator on level-1 partitions. If PART is displayed, dynamic pruning is used. For example, Iterations: 2 in example 3 indicates that the iteration operator needs to traverse two level-1 partitions. Iterations: PART indicates that the number of level-1 partitions to be traversed is determined by parameter conditions of the partition key.
Sub Iterations	Specifies the number of iterations performed by the partition iteration operator on level-2 partitions. If PART is displayed, dynamic pruning is used. For example, Sub Iterations: 2 in example 3 indicates that the iteration operator needs to traverse two level-2 partitions. Iterations: PART indicates that the number of level-2 partitions to be traversed is determined by parameter conditions of the partition key.

Item	Description
Filter	Filter predicate of the operator. In the example 2, the filter condition is that the value in column category is less than or equal to 5 and the value in column id is equal to 0. When a query is executed, rows that meet these conditions are included in the final result set.
Partitioned Seq Scan	Scan mode of a partitioned table.
Selected Partitions	Level-1 partition pruning result. In example 3, Selected Partitions: 1..2 indicates that partitions 1 and 2 are selected.
Selected Subpartitions	Level-2 partition pruning result. In example 3, Selected Subpartitions: 1:1 2:1 indicates that subpartition 1 of the first level-1 partition and subpartition 1 of the second level-1 partition are selected. Selected Subpartitions: ALL indicates that all level-2 partitions are selected.

6.2.3.7 Other Keywords

6.2.3.7.1 InitPlan and Subplan

Description

InitPlan is a part of the GaussDB subplan. In GaussDB, subquery plans can be classified into related subplans and non-related subplans. A related subplan is a row that a subquery depends on an external query and cannot be executed independently of the foreign query. A non-related subplan is the opposite. In GaussDB, a SubPlan or InitPlan can be called a subplan. It is a part that can be independently executed compared with the entire plan and is generally generated by a subplan that cannot be promoted. SubPlans are generated based on correlated subplans, and InitPlans are generated based on non-related subplans. SubPlan runs during the execution of the main query and is re-executed on each row of the main query. InitPlan runs before the execution of the main query and the result is one-time. They are calculated at the beginning of the query and then cached for reuse during the entire query execution, therefore, InitPlan is more efficient.

Typical Scenarios

The SELECT subplan does not depend on foreign queries and cannot be pulled up.

Examples

Example: The SELECT subplan does not depend on foreign queries and cannot be pulled up.

```
-- Prepare data.
gaussdb=# DROP TABLE IF EXISTS init_table;
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
gaussdb=# CREATE TABLE init_table (id int, grade int, time int);
CREATE TABLE
gaussdb=# INSERT INTO init_table VALUES(generate_series(1,10000), (random() * 10)::integer,(random() *
10)::integer );
INSERT 0 10000
gaussdb=# CREATE TABLE t1(grade int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES( 10),( 11);
INSERT 0 2
gaussdb=# CREATE TABLE t2(a int, b int, c int);
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES(1, 2, 3 );
INSERT 0 1

-- Execution result.
gaussdb=# EXPLAIN SELECT * FROM init_table WHERE init_table.grade IN (SELECT grade FROM t2) AND
init_table.grade != (SELECT * FROM t1);
QUERY PLAN
-----
Streaming (type: GATHER) (cost=13.56..126.53 rows=9 width=12)
Node/s: All datanodes
InitPlan 2 (returns $2)
-> Streaming(type: BROADCAST) (cost=0.00..13.18 rows=40 width=4)
    Spawn on: All datanodes
    -> Seq Scan on t1 (cost=0.00..13.13 rows=20 width=4)
-> Seq Scan on init_table (cost=0.00..112.79 rows=9 width=12)
    Filter: ((grade <> $2) AND (SubPlan 1))
    SubPlan 1
    -> Result (cost=0.00..15.28 rows=40 width=0)
        -> Materialize (cost=0.00..15.28 rows=40 width=0)
            -> Streaming(type: BROADCAST) (cost=0.00..15.18 rows=40 width=0)
                Spawn on: All datanodes
            -> Seq Scan on t2 (cost=0.00..13.13 rows=20 width=0)

(14 rows)

-- Drop.
gaussdb=# DROP TABLE IF EXISTS init_table;
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# DROP TABLE IF EXISTS t2;
```

In the preceding example, the output is as follows.

Item	Description
InitPlan	Non-correlated subplan. In the example, InitPlan 2 indicates that subplan 2 is a non-correlated subplan.
returns	Returned result. In the example, returns \$2 indicates that the result of InitPlan is stored in \$2 .
SubPlan	Correlated subplan. In the example, SubPlan 1 indicates that subplan 1 is a non-correlated subplan.

6.3 Optimization Process

You can analyze slow SQL statements to optimize them.

Procedure

- Step 1** Collect all table statistics associated with the SQL statements. In a database, statistics indicate the source data of a plan generated by an optimizer. If no collection statistics are available or out of date, the execution plan may seriously deteriorate, leading to low performance. According to past experience, about 10% performance problems occurred because no statistics are collected. For details, see [Updating Statistics](#).
- Step 2** View the execution plan to find out the cause. If the SQL statements have been running for a long period of time and not ended, run the **EXPLAIN** command to view the execution plan and then locate the fault. If the SQL statement has been properly executed, execute EXPLAIN ANALYZE or EXPLAIN PERFORMANCE to check the execution plan and information to locate the fault. For details about the execution plan, see [Introduction to the SQL Execution Plan](#).
- Step 3** [Review and modify a table definition](#).
- Step 4** For details about EXPLAIN or EXPLAIN PERFORMANCE, the reason why SQL statements are slowly located, and how to solve this problem, see [Typical SQL Optimization Methods](#).
- Step 5** Generally, some SQL statements can be converted to its equivalent statements in all or certain scenarios by rewriting queries. SQL statements are simpler after they are rewritten. Some execution steps can be simplified to improve the performance. Query rewriting methods are universal in all databases. [Experience in Rewriting SQL Statements](#) describes several tuning methods by rewriting SQL statements.

----End

6.4 Updating Statistics

In a database, statistics indicate the source data of a plan generated by an optimizer. If no statistics are available or out of date, the execution plan may seriously deteriorate, leading to low performance.

Context

The ANALYZE statement collects statistic about table contents in databases, which will be stored in the PG_STATISTIC system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

After executing batch insertions and deletions, you are advised to execute the ANALYZE statement on the table or the entire library to update statistics. By default, 30,000 rows of statistics are sampled. That is, the default value of the GUC parameter **default_statistics_target** is **100**. If the total number of rows in the table exceeds 1,600,000, you are advised to set **default_statistics_target** to **-2**, indicating that 2% of the statistics are collected.

For an intermediate table generated during the execution of a batch script or stored procedure, you also need to run the **ANALYZE** statement.

If there are multiple inter-related columns in a table and the conditions or grouping operations based on these columns are involved in the query, collect statistics about these columns so that the query optimizer can accurately estimate the number of rows and generate an effective execution plan.

If the table has a GSI, perform ANALYZE on the base table first and then on the GSI.

Procedure

Run the following commands to update the statistics about a table or the entire database:

```
ANALYZE tablename;           -- Update statistics about a table.  
ANALYZE;                       -- Update statistics about the entire database.
```

Run the following command to update statistics about the GSI on the table:

```
ANALYZE GLOBAL INDEX indexname FOR TABLE tablename;
```

NOTE

To update the statistics about the GSI, perform ANALYZE on the base table first.

Run the following statements to perform statistics-related operations on multiple columns:

```
ANALYZE tablename ((column_1, column_2));           -- Collect statistics about column_1 and  
column_2 of tablename.  
  
ALTER TABLE tablename ADD STATISTICS ((column_1, column_2)); -- Declare statistics about column_1  
and column_2 of tablename.  
ANALYZE tablename;                                     -- Collect statistics about one or more columns.  
  
ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2)); --Delete statistics about column_1  
and column_2 of tablename or their statistics declaration.
```

NOTICE

After the statistics are declared for multiple columns by executing the ALTER TABLE *tablename* ADD STATISTICS statement, the system collects the statistics about these columns next time ANALYZE is performed on the table or the entire database.

To collect the statistics, run the **ANALYZE** command.

NOTE

Use EXPLAIN to show the execution plan of each SQL statement. If **rows** is set to **10** (the default value, probably indicating that the table has not been analyzed) is displayed in the SEQ SCAN output of a table, execute the ANALYZE statement for this table.

6.5 Reviewing and Modifying a Table Definition

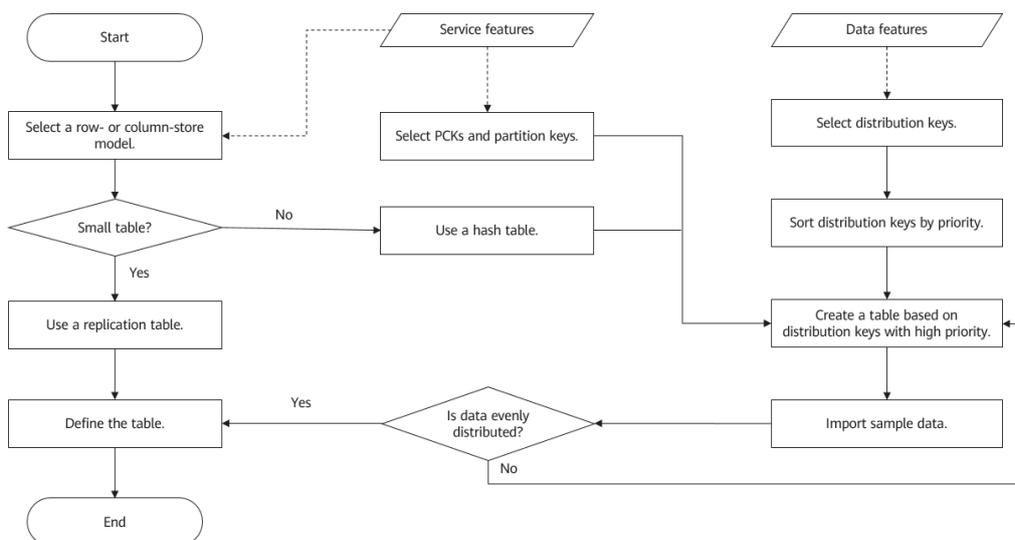
6.5.1 Overview

In a distributed framework, data is distributed on DNs. Data on one or more DNs is stored on a physical storage device. To properly define a table, you must:

1. **Evenly distribute data on each DN** to avoid the available capacity decrease of a cluster caused by insufficient storage space of the storage device associated with a DN. Select a proper distribution key to avoid data skew.
2. **Evenly assign table scanning tasks on each DN** to avoid that a single DN is overloaded by the table scanning tasks. Specifically, do not select columns in the equivalent filter of a base table as the distribution key.
3. **Reduce the data volume scanned** by using the partition pruning mechanism.
4. **Minimize random I/Os** by using clustering.
5. **Avoid data shuffle** to reduce the network pressure by selecting the **join-condition** or **group by** column as the distribution key.

The distribution key is the core for defining a table. [Figure 6-2](#) shows the procedure of defining a table. The table definition is created during the database design and is reviewed and modified during the SQL statement optimization.

Figure 6-2 Procedure of defining a table



6.5.2 Selecting a Distribution Mode

In replication mode, full data in a table is copied to each DN in the cluster. This mode is used for tables containing a small volume of data. Full data in a table stored on each each DN avoids data redistribution during the join operation. This reduces network costs and plan segment (each having a thread), but generates much redundant data. Generally, this mode is only used for small dimension tables.

In hash mode, hash values are generated for one or more columns. You can obtain the storage location of a tuple based on the mapping between DNs and the hash values. In a hash table, I/O resources on each node can be used during data read/

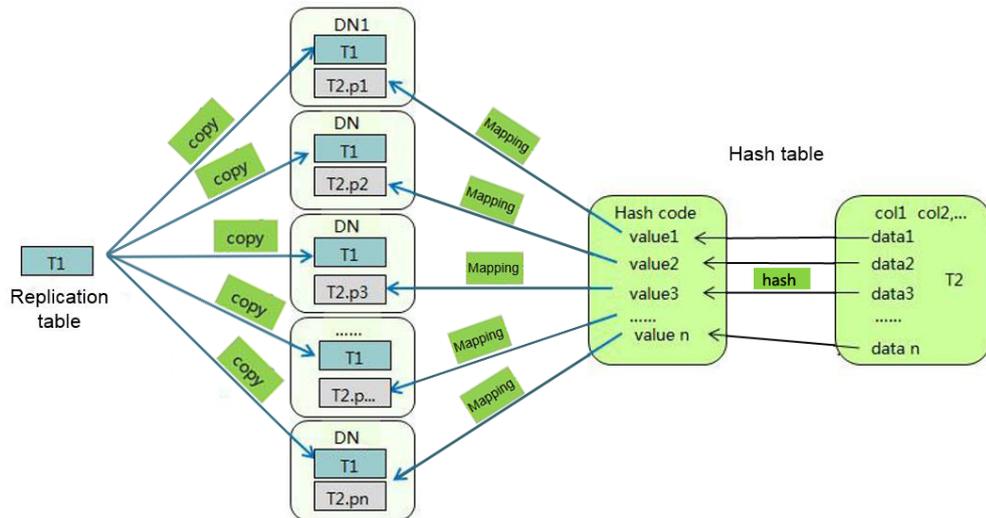
write, which improves the read/write speed of a table. Generally, a table containing a large amount data is defined as a hash table.

Range distribution and list distribution are user-defined distribution policies. Values in a distribution key are within a certain range or fall into a specific value range of the corresponding target DN. The two distribution modes facilitate flexible data management which, however, requires users equipped with certain data abstraction capability.

Policy	Description	Application Scenario
Hash	Table data is distributed on all DNs in the cluster.	Fact tables containing a large amount of data
Replication	Full data in a table is stored on every DN in the cluster.	Small tables and dimension tables
Range	Table data is mapped to specified columns based on the range and distributed to the corresponding DNs.	Users need to customize distribution rules.
List	Table data is mapped to specified columns based on specific values and distributed to corresponding DNs.	Users need to customize distribution rules.

As shown in **Figure 6-3**, T1 is a replication table and T2 is a hash table.

Figure 6-3 Replication tables and hash tables



NOTE

- When you insert, modify, or delete data in a replication table, if you use the shippable or immutable function to encapsulate components that cannot be pushed down, data on different DNs in the replication table may be inconsistent.
- If statements with unstable results, such as window functions, rownum, and limit clauses and user-defined functions, are used to insert or modify data in a replication table, data on different nodes may be different.

6.5.3 Selecting Distribution Keys

Selecting a distribution key for a hash table is essential. Details are as follows:

1. **Ensure that the column values are discrete so that data can be evenly distributed to each DN.** You can select the primary key of the table as the distribution key. For example, for a person information table, choose the ID card number column as the distribution key.
2. **Do not select the column that has a constant filter.** For example, if a constant constraint (for example, `zqdh='000001'`) exists on the `zqdh` column in some queries on the `dwcjck` table, you are not advised to use `zqdh` as the distribution key.
3. **Select the join condition as the distribution key,** so that join tasks can be pushed down to DNs to execute, reducing the amount of data transferred between the DNs.

For a hash table, an improper distribution key may cause data skew or poor I/O performance on certain DNs. Therefore, you need to check the table to ensure that data is evenly distributed on each DN. You can run the following SQL statement to check for data skew:

```
select
xc_node_id, count(1)
from tablename
group by xc_node_id
order by xc_node_id desc;
```

`xc_node_id` corresponds to a DN. Generally, **over 5% difference between the amount of data on different DNs is regarded as data skew. If the difference is over 10%, choose another distribution key.**

Multiple distribution keys can be selected in GaussDB to evenly distribute data.

You can select the distribution key of the range or list distribution table as required. In addition to selecting a proper distribution key, pay attention to the impact of distribution rules on data distribution.

6.5.4 Using Partitioned Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored on these physical partitions, instead of the logical partitioned table. A partitioned table has the following advantages over an ordinary table:

1. High query performance: You can specify partitions when querying partitioned tables, improving query efficiency.
2. High availability: If a certain partition in a partitioned table is faulty, data in the other partitions is still available.
3. Easy maintenance: To fix a partitioned table having a faulty partition, you only need to fix the partition.

GaussDB supports range partitioned tables, list partitioned tables, and hash partitioned tables.

- Range partitioned table: Data in different ranges is mapped to different partitions. The range is determined by the partition key specified during the

partitioned table creation. The partition key is usually a date. For example, sales data is partitioned by month.

- List partitioned table: Key values contained in the data are stored in different partitions, and the data is mapped to each partition in sequence. The key values contained in the partitions are specified when the partitioned table is created.
- Hash partitioned table: Data is mapped to each partition based on the internal hash algorithm. The number of partitions is specified when the partitioned table is created.

6.5.5 Selecting a Data Type

Use the following principles to select efficient data types:

1. Select data types that facilitate data calculation.

Generally, the calculation of integers (including common comparison calculations, such as =, >, <, ≥, ≤, and ≠ and GROUP BY) is more efficient than that of strings and floating point numbers.

2. Select data types with a short length.

Data types with short length reduce both the data file size and the memory used for computing, improving the I/O and computing performance. For example, use **SMALLINT** instead of **INT**, and **INT** instead of **BIGINT**.

3. Use the same data type for a join.

You are advised to use the same data type for a join. To join columns with different data types, the database needs to convert them to the same type, which leads to additional performance overheads.

6.6 Reviewing the Plan Change Scenario

The optimizer generates the optimal execution plan based on the cost model and statistics of related tables. Therefore, changes in statistics of related tables or parameter modifications of the cost model affect generation of the optimal plan. In addition, many operations of production services may cause the plan cache to become invalid and rebuild, such as operations related to statistics update (VACUUM, ANALYZE, AUTOVACUUM, and AUTOANALYZE), DDL operations such as index rebuilding, SQL patch application, and other scenarios that cause the cache to become invalid. If the optimal plan generated based on the cost during the period is different from the previous one, a plan jump occurs. Therefore, plan change is a common phenomenon in production services.

Generally, positive plan jumps do not need to be concerned. Pay special attention to the scenarios where the performance deteriorates and the CPU usage increases due to plan changes. This is usually accompanied by database environment variable changes, such as statistics update, table structure change, or database version upgrade.

In GaussDB, plan changes may occur in the following scenarios. Therefore, you need to review the scenarios and apply precise tuning methods (such as hints).

- Generic plan rebuilding: When the service is running stably, the generic plan is usually executed, indicating that the plan is stable and the optimizer does not

need to optimize the SQL statement. In some cases, the generic plan may be invalid and needs to be rebuilt.

- If you modify GUC parameters, the generic plan will be rebuilt. If you modify optimizer parameters, such as **query_dop**, plan changes will occur. Do not modify optimizer parameters unless there are special requirements for stable services.
- DDL operations are performed on the base table involved in the query. The DDL operations on the table cause the rebuilding of the generic plan, and the addition, deletion, and modification of indexes affect the plan selection. To optimize an index, ensure that the index is valid and update the index statistics in a timely manner to select a better plan.
- Perform IUD operations on tables and manually or automatically collect statistics. Changes in statistics cause generic plan rebuilding. Generally, if statistics are collected in a timely manner, performance deterioration will not occur even if the plan changes. A common service scenario is that a large number of IUD operations are performed on the base table involved in the query but the statistics are not updated in a timely manner. In this case, you need to manually collect statistics or review the triggering of automatic statistics collection.
- Intelligent optimizer: The generic plan cannot meet service requirements in some scenarios. For example, in data skew scenarios, the optimal plan varies depending on data. The intelligent optimizer generates multiple alternative generic plans and selects the optimal plan for execution in real time based on the query feedback. The intelligent optimizer is related to the service execution sequence. If the intelligent optimizer cannot meet the requirements, you are advised to use hints or SQL patches to optimize key service statements.

6.7 Typical SQL Optimization Methods

SQL optimization involves continuous analysis and trying. Queries are run before they are used for services to determine whether the performance meets requirements. If it does not, queries will be optimized by [checking the execution plan](#) and identifying the causes. Then, the queries will be run and optimized again until they meet the requirements.

6.7.1 Optimizing SQL Self-diagnosis

Performance issues may occur when you query data or run the **INSERT**, **DELETE**, **UPDATE**, or **CREATE TABLE AS** statement.

Alarms that can trigger SQL self diagnosis depend on the settings of the GUC parameter **resource_track_level**. If **resource_track_level** is set to **query**, alarms about the failures in collecting column statistics and pushing down SQL statements will trigger the diagnosis. If **resource_track_level** is set to **operator**, all alarms will trigger the diagnosis.

Whether an SQL plan will be diagnosed depends on the settings of the GUC parameter **resource_track_cost**. An SQL plan will be diagnosed only if its execution cost is greater than **resource_track_cost**. You can use the **EXPLAIN** keyword to check the plan execution cost.

The SQL self-diagnosis function is affected by the **enable_analyze_check** parameter. Ensure that the function is enabled before using it.

If a large number of statements are executed, certain data may fail to be collected due to memory control. In this case, you can increase the value of **instr_unique_sql_count**.

Alarms

Currently, the following performance alarms will be reported:

- Some column statistics are not collected.

An alarm will be reported if some column statistics are not collected.

Example alarms:

No statistics about a table are not collected.

```
Statistic Not Collect:  
schema_test.t1
```

The statistics about a single column are not collected.

```
Statistic Not Collect:  
schema_test.t2(c1,c2)
```

The statistics about multiple columns are not collected.

```
Statistic Not Collect:  
schema_test.t3((c1,c2))
```

The statistics about a single column and multiple columns are not collected.

```
Statistic Not Collect:  
schema_test.t4(c1,c2) schema_test.t4((c1,c2))
```

- SQL statements are not pushed down.

The cause details are displayed in the alarms. For details about the optimization, see [Optimizing Statement Pushdown](#).

- If a function is not pushed down, an alarm is generated for the corresponding function.
- If a syntax does not support pushdown, an alarm is generated, indicating that the syntax does not support pushdown.

Example alarms:

```
SQL is not plan-shipping, reason : "With Recursive" can not be shipped"  
SQL is not plan-shipping, reason : "Function now() can not be shipped"  
SQL is not plan-shipping, reason : "Function string_agg() can not be shipped"
```

- In a hash join, the larger table is used as the inner table.

An alarm will be reported if the number of rows in the inner table reaches or exceeds 10 times of that in the outer table, more than 100,000 inner-table rows are processed on each DN in average, and the join statement has spilled to disks. You can check whether a hash join is used. For details about the optimization, see [Hint-based Tuning](#).

Example alarms:

```
PlanNode[7] Large Table is INNER in HashJoin "Hash Aggregate"
```

- **nestloop** is used in a large-table equivalent join.

An alarm will be reported if **nestloop** is used in an equivalent join where more than 100,000 larger-table rows are processed on each DN in average. You can check whether **nestloop** is used. For details about the optimization, see [Hint-based Tuning](#).

Example alarms:

```
PlanNode[5] Large Table with Equal-Condition use Nestloop"Nested Loop"
```

- A large table is broadcasted.

An alarm will be reported if more than 100 thousand of rows are broadcast on each DN in average. For details about the optimization, see [Hint-based Tuning](#).

Example alarms:

```
PlanNode[5] Large Table in Broadcast "Streaming(type: BROADCAST dop: 1/2)"
```

- Data skew occurs.

An alarm will be reported if the number of rows processed on any DN exceeds 100 thousand, and the number of rows processed on a DN reaches or exceeds 10 times of that processed on another DN.

Example alarms:

```
PlanNode[6] DataSkew:"Seq Scan", min_dn_tuples:0, max_dn_tuples:524288
```

- Estimation is inaccurate.

An alarm will be reported if the number of rows estimated by the optimizer is greater than 100,000 compared with the actual number of rows and the estimated number of rows is 10 times or more than the actual number of rows. For details about the optimization, see [Hint-based Tuning](#).

Example alarms:

```
PlanNode[5] Inaccurate Estimation-Rows: "Hash Join" A-Rows:0, E-Rows:52488
```

Restrictions

1. An alarm contains a maximum of 2048. If the length of an alarm exceeds this value (for example, a large number of long table names and column names are displayed in the alarm when their statistics are not collected), a warning instead of an alarm will be reported.

```
WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"
```
2. If a query statement contains the **Limit** operator, alarms of operators lower than **Limit** will not be reported.
3. For alarms about data skew and inaccurate estimation, only alarms on the lower-layer nodes in a plan tree will be reported. This is because the same alarms on the upper-level nodes may be triggered by problems on the lower-layer nodes. For example, if data skew occurs on the **Scan** node, data skew may also occur in operators (for example, **Hashagg**) at the upper layer.

6.7.2 Optimizing Statement Pushdown

Statement Pushdown

Currently, the GaussDB optimizer can use three methods to develop statement execution policies in the distributed framework: generating a statement pushdown plan, a distributed execution plan, or a distributed execution plan for sending statements.

- A statement pushdown plan pushes query statements from a CN down to DNs for execution and returns the execution results to the CN.
- In a distributed execution plan, a CN compiles and optimizes query statements, generates a plan tree, and then sends the plan tree to DNs for execution. After the statements have been executed, execution results will be returned to the CN.
- A distributed execution plan for sending statements pushes queries that can be pushed down (mostly base table scanning statements) to DNs for execution. Then, the plan obtains the intermediate results and sends them to the CN, on which the remaining queries are to be executed.

The third policy sends many intermediate results from DNs to the CN for further execution. In this case, the CN performance bottleneck (in bandwidth, storage, and computing) is caused by statements that cannot be pushed down to DNs. Therefore, you are advised not to use the query statements where only the third policy applies.

Statements cannot be pushed down if they have **functions that do not support pushdown** or **syntax that does not support pushdown**. Generally, you can rewrite the execution statements to solve the problem.

Typical Scenarios of Statement Pushdown

In the GaussDB optimizer, if you want to support statement pushdown, set the GUC parameter **enable_fast_query_shipping** to **on**. Generally, no execution plan operator is displayed after the **EXPLAIN** statement. If the keyword similar to "Data Node Scan on" exists, the statement has been pushed down to DNs for execution. The following describes statement pushdown and its supported scope from three scenarios.

1. Pushdown of single-table query statements

In a distributed database, to query a single table, whether the current statement can be pushed down depends on whether the CN needs to participate in calculation instead of simply collecting data. If the CN needs to further calculate the DN result, the statement cannot be pushed down. Generally, statements with keywords such as agg, windows function, limit/offset, sort, distinct cannot be pushed down.

- Pushdown: Simple queries can be pushed down without further calculation on the CN.

```
gaussdb=# explain select * from t where c1 > 1;
          QUERY PLAN
-----
Data Node Scan on "__REMOTE_FQS_QUERY__" (cost=0.00..0.00 rows=0 width=0)
Node/s: All datanodes
(2 rows)
```

- Non-pushdown: A CN with the **limit** clause cannot simply send statements to DNs and collect data, which is inconsistent with the semantics of the **limit** clause.

```
gaussdb=# explain select * from t limit 1;
                QUERY PLAN
-----
Limit (cost=0.00..0.00 rows=1 width=12)
-> Data Node Scan on "_REMOTE_LIMIT_QUERY_" (cost=0.00..0.00 rows=1 width=12)
    Node/s: All datanodes
(3 rows)
```

- Non-pushdown: A CN with the aggregate function cannot simply push down statements. Instead, it needs to further aggregate the results collected from DNs.

```
gaussdb=# explain select sum(c1), count(*) from t;
                QUERY PLAN
-----
Aggregate (cost=0.10..0.11 rows=1 width=20)
-> Data Node Scan on "_REMOTE_GROUP_QUERY_" (cost=0.00..0.00 rows=20 width=4)
    Node/s: All datanodes
(3 rows)
```

2. Pushdown of multi-table query statements

In the multi-table query scenario, whether a statement can be pushed down depends on the **join** condition and distribution columns. That is, if the **join** condition matches the distribution columns of the table, the statement can be pushed down. Otherwise, the statement cannot be pushed down. Generally, a replication table can be pushed down.

- Create two hash distribution tables.

```
gaussdb=# create table t(c1 int, c2 int, c3 int)distribute by hash(c1);
CREATE TABLE
gaussdb=# create table t1(c1 int, c2 int, c3 int)distribute by hash(c1);
CREATE TABLE
```

- Pushdown: The **join** condition meets the hash distribution key attributes of two tables.

```
gaussdb=# explain select * from t1 join t on t.c1 = t1.c1;
                QUERY PLAN
-----
Data Node Scan on "_REMOTE_FQS_QUERY_" (cost=0.00..0.00 rows=0 width=0)
    Node/s: All datanodes
(2 rows)
```

- Non-pushdown: The **join** condition does not meet the hash distribution key attribute. That is, **t1.c2** is not the distribution key of **t1**.

```
gaussdb=# explain select * from t1 join t on t.c1 = t1.c2;
                QUERY PLAN
-----
Hash Join (cost=0.25..0.53 rows=20 width=24)
  Hash Cond: (t1.c2 = t.c1)
-> Data Node Scan on t1 "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=20 width=12)
    Node/s: All datanodes
-> Hash (cost=0.00..0.00 rows=20 width=12)
    -> Data Node Scan on t "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=20 width=12)
        Node/s: All datanodes
(7 rows)
```

- Delete two hash distribution tables.

```
gaussdb=# DROP TABLE t;
DROP TABLE
gaussdb=# DROP TABLE t1;
DROP TABLE
```

3. Special scenarios

In some special scenarios, for example, a statement containing the WITH RECURSIVE clause cannot be pushed down.

Checking Whether the Execution Plan Has Been Pushed Down

Perform the following procedure to quickly determine whether the execution plan can be pushed down:

Step 1 Set the GUC parameter **enable_fast_query_shipping** to **off** to use the distributed framework policy for the query optimizer.

```
SET enable_fast_query_shipping = off;
```

Step 2 View the execution plan.

If the execution plan contains Data Node Scan nodes, the SQL statements cannot be pushed down to DNs. If the execution plan contains Streaming nodes, the SQL statements can be pushed down to DNs.

For example:

```
gaussdb=# select  
count(ss.ss_sold_date_sk order by ss.ss_sold_date_sk)c1  
from store_sales ss, store_returns sr  
where  
sr.sr_customer_sk = ss.ss_customer_sk;
```

The execution plan is as follows, which indicates that the SQL statement cannot be pushed down.

```
-----  
QUERY PLAN  
-----  
Aggregate  
-> Hash Join  
Hash Cond: (ss.ss_customer_sk = sr.sr_customer_sk)  
-> Data Node Scan on store_sales "_REMOTE_TABLE_QUERY_"  
Node/s: All datanodes  
-> Hash  
-> Data Node Scan on store_returns "_REMOTE_TABLE_QUERY_"  
Node/s: All datanodes  
(8 rows)
```

----End

Syntax That Does Not Support Pushdown

SQL syntax that does not support pushdown is described using the following table definition examples:

```
gaussdb=# CREATE TABLE CUSTOMER1  
(  
  C_CUSTKEY    BIGINT NOT NULL  
  , C_NAME     VARCHAR(25) NOT NULL  
  , C_ADDRESS  VARCHAR(40) NOT NULL  
  , C_NATIONKEY INT NOT NULL  
  , C_PHONE    CHAR(15) NOT NULL  
  , C_ACCTBAL  DECIMAL(15,2) NOT NULL  
  , C_MKTSEGMENT CHAR(10) NOT NULL  
  , C_COMMENT  VARCHAR(117) NOT NULL  
)  
DISTRIBUTE BY hash(C_CUSTKEY);  
gaussdb=# CREATE TABLE test_stream(a int, b float); -- float does not support redistribution.  
gaussdb=# CREATE TABLE sal_emp ( c1 integer[] ) DISTRIBUTE BY replication;
```

- The **returning** statement cannot be pushed down.

```

gaussdb=# explain update customer1 set C_NAME = 'a' returning c_name;
QUERY PLAN
-----
Update on customer1 (cost=0.00..0.00 rows=30 width=187)
Node/s: All datanodes
Node expr: c_custkey
-> Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30 width=187)
Node/s: All datanodes
(5 rows)

```
- If an SQL statement contains the aggregate functions using **ORDER BY**, this statement cannot be pushed down.

```

gaussdb=# explain verbose select count ( c_custkey order by c_custkey) from customer1;
QUERY PLAN
-----
Aggregate (cost=2.50..2.51 rows=1 width=8)
Output: count(customer1.c_custkey ORDER BY customer1.c_custkey)
-> Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30 width=8)
Output: customer1.c_custkey
Node/s: All datanodes
Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
(6 rows)

```
- If an SQL statement contains **COUNT(DISTINCT expr)** and columns in **COUNT(DISTINCT expr)** do not support redistribution, this statement cannot be pushed down.

```

gaussdb=# explain verbose select count(distinct b) from test_stream;
QUERY PLAN
-----
Aggregate (cost=2.50..2.51 rows=1 width=8)
Output: count(DISTINCT test_stream.b)
-> Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30 width=8)
Output: test_stream.b
Node/s: All datanodes
Remote query: SELECT b FROM ONLY public.test_stream WHERE true
(6 rows)

```
- A statement containing **DISTINCT ON** cannot be pushed down.

```

gaussdb=# explain verbose select distinct on (c_custkey) c_custkey from customer1 order by c_custkey;
QUERY PLAN
-----
Unique (cost=49.83..54.83 rows=30 width=8)
Output: customer1.c_custkey
-> Sort (cost=49.83..52.33 rows=30 width=8)
Output: customer1.c_custkey
Sort Key: customer1.c_custkey
-> Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30
width=8)
Output: customer1.c_custkey
Node/s: All datanodes
Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
(9 rows)

```
- A statement containing array expressions cannot be pushed down.

```

gaussdb=# explain verbose select array[c_custkey,1] from customer1 order by c_custkey;
QUERY PLAN
-----
Sort (cost=49.83..52.33 rows=30 width=8)
Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey
Sort Key: customer1.c_custkey
-> Data Node Scan on "_REMOTE_SORT_QUERY_" (cost=0.00..0.00 rows=30 width=8)
Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey
Node/s: All datanodes
Remote query: SELECT ARRAY[c_custkey, 1::bigint], c_custkey FROM ONLY public.customer1
WHERE true ORDER BY 2
(7 rows)

```
- The following table describes the scenarios where a statement containing **WITH RECURSIVE** cannot be pushed down in the current version, as well as the causes.

No.	Scenario	Cause of Not Supporting Pushdown
1	The query contains foreign tables.	<p>LOG: SQL can't be shipped, reason: RecursiveUnion contains ForeignScan is not shippable (In this table, LOG describes the cause of not supporting pushdown.)</p> <p>In the current version, queries containing foreign tables do not support pushdown.</p>
2	Multiple node groups exist.	<p>LOG: SQL can't be shipped, reason: With-Recursive under multi-nodegroup scenario is not shippable</p> <p>In the current version, pushdown is supported only when all base tables are stored and computed in the same node group.</p>
3	UNION does not contain ALL, and deduplication is required.	<p>LOG: SQL can't be shipped, reason: With-Recursive does not contain "ALL" to bind recursive & none-recursive branches</p> <p>For example:</p> <pre>WITH recursive t_result AS (SELECT dm,sj_dm,name,1 as level FROM test_rec_part WHERE sj_dm > 10 UNION SELECT t2.dm,t2.sj_dm,t2.name ' > ' t1.name,t1.level+1 FROM t_result t1 JOIN test_rec_part t2 ON t2.sj_dm = t1.dm) SELECT * FROM t_result t;</pre>

No.	Scenario	Cause of Not Supporting Pushdown
4	A base table contains a system catalog.	<p>LOG: SQL can't be shipped, reason: With-Recursive contains system table is not shippable</p> <p>Example:</p> <pre>WITH RECURSIVE x(id) AS (select count(1) from pg_class where oid=1247 UNION ALL SELECT id+1 FROM x WHERE id < 5), y(id) AS (select count(1) from pg_class where oid=1247 UNION ALL SELECT id+1 FROM x WHERE id < 10) SELECT y.*, x.* FROM y LEFT JOIN x USING (id) ORDER BY 1;</pre>
5	Only VALUES is used for scanning base tables. In this case, the statement can be executed on the CN, and DNs are unnecessary.	<p>LOG: SQL can't be shipped, reason: With-Recursive contains only values rte is not shippable</p> <p>Example:</p> <pre>WITH RECURSIVE t(n) AS (VALUES (1) UNION ALL SELECT n+1 FROM t WHERE n < 100) SELECT sum(n) FROM t;</pre>
6	Only the recursion part has correlation conditions of correlated subqueries, and the non-recursion part has no correlation condition.	<p>LOG: SQL can't be shipped, reason: With-Recursive recursive term correlated only is not shippable</p> <p>Example:</p> <pre>select a.ID,a.Name, (with recursive cte as (select ID, PID, NAME from b where b.ID = 1 union all select parent.ID,parent.PID,parent.NAME from cte as child join b as parent on child.pid=parent.id where child.ID = a.ID) select NAME from cte limit 1) cName from (select id, name, count(*) as cnt from a group by id,name) a order by 1,2;</pre>

No.	Scenario	Cause of Not Supporting Pushdown
7	The replicate plan is used for limit in the non-recursion part but the hash plan is used in the recursion part, resulting in conflicts.	<p>LOG: SQL can't be shipped, reason: With-Recursive contains conflict distribution in none-recursive(Replicate) recursive(Hash)</p> <p>Example:</p> <pre>WITH recursive t_result AS (select * from(SELECT dm,sj_dm,name,1 as level FROM test_rec_part WHERE sj_dm < 10 order by dm limit 6 offset 2) UNION all SELECT t2.dm,t2.sj_dm,t2.name ' > ' t1.name,t1.level+1 FROM t_result t1 JOIN test_rec_part t2 ON t2.sj_dm = t1.dm) SELECT * FROM t_result t;</pre>
8	recursive of multiple-layers are nested. That is, a recursive is nested in the recursion part of another recursive .	<p>LOG: SQL can't be shipped, reason: Recursive CTE references recursive CTE "cte"</p> <p>Example:</p> <pre>with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from (with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from rec_tb4 h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID) h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID,1,2,3;</pre>

Delete the tables.

```
gaussdb=# DROP TABLE CUSTOMER1;
DROP TABLE
gaussdb=# DROP TABLE test_stream;
DROP TABLE
gaussdb=# DROP TABLE saL_emp;
DROP TABLE
```

Functions That Do Not Support Pushdown

The following describes the volatility of functions. In GaussDB, every function has a volatility classification, with the possibilities being:

- **IMMUTABLE**
Indicates that the function always returns the same result if the parameter values are the same.
- **STABLE**
Indicates that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.
- **VOLATILE**
Indicates that the function value can change in a single table scan and no optimization is performed.

The volatility of a function can be obtained by querying for its **provolatile** column in **pg_proc**. The value **i** indicates immutable, **s** indicates stable, and **v** indicates volatile. The valid values of the **proshippable** column in **pg_proc** are **t**, **f**, and **NULL**. This column and the **provolatile** column together describe whether a function is pushed down.

- If the **provolatile** of a function is **i**, the function can be pushed down regardless of the value of **proshippable**.
- If the **provolatile** of a function is **s** or **v**, the function can be pushed only if the value of **proshippable** is **t**.
- CTEs containing **random**, **exec_hadoop_sql**, or **exec_on_extension** are not pushed down, because pushdown may lead to incorrect results.

When creating a user-defined function, you can specify the values of **provolatile** and **proshippable**. For details, see [CREATE FUNCTION](#).

In scenarios where a function does not support pushdown, perform one of the following as required:

- If it is a system function, replace it with a functionally equivalent one.
- If it is a user-defined function, check whether its **provolatile** and **proshippable** are correctly defined.

Example: User-defined Functions

Define a user-defined function that generates fixed output for a certain input as the **immutable** type.

Take the sales information of TPC Benchmark DS (TPC-DS) as an example. If you want to write a function to calculate the discount data of a product, you can define the function as follows:

```
CREATE FUNCTION func_percent_2 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
VOLATILE;
```

Run the following statements:

```
SELECT func_percent_2(ss_sales_price, ss_list_price)
FROM store_sales;
```

The execution plan is as follows.

```
Data Node Scan on store_sales " _REMOTE_TABLE_QUERY_ "  
  Output: func_percent_2(store_sales.ss_sales_price, store_sales.ss_list_price)  
  Remote query: SELECT ss_sales_price, ss_list_price FROM ONLY store_sales WHERE true  
(3 rows)
```

func_percent_2 is not pushed down, and **ss_sales_price** and **ss_list_price** are executed on a CN. In this case, a large amount of resources on the CN is consumed, and the performance deteriorates as a result.

In this example, the function generates the same output when the same input is provided. Therefore, we can modify the function to the following one:

```
CREATE FUNCTION func_percent_1 (NUMERIC, NUMERIC) RETURNS NUMERIC  
AS 'SELECT $1 / $2 WHERE $2 > 0.01'  
LANGUAGE SQL  
IMMUTABLE;
```

Run the following statement:

```
SELECT func_percent_1(ss_sales_price, ss_list_price)  
FROM store_sales;
```

The execution plan is as follows.

```
Data Node Scan  
  Output: (func_percent_1(store_sales.ss_sales_price, store_sales.ss_list_price))  
  Remote query: SELECT func_percent_1(ss_sales_price, ss_list_price) AS func_percent_1 FROM store_sales  
(3 rows)
```

The **func_percent_1** function is pushed down to DNs for execution.

6.7.3 Optimizing Subqueries

Context

When an application runs an SQL statement to operate the database, a large number of subqueries are used because they are more clear than table join. Especially in complicated query statements, subqueries have more complete and independent semantics, which makes SQL statements clearer and easier to understand. Therefore, subqueries are widely used.

In GaussDB, subqueries can also be called sublinks based on the location of subqueries in SQL statements.

- Subquery: corresponds to a range table (RangeTblEntry) in the query parse tree. That is, a subquery is a SELECT statement following immediately after the FROM keyword.
- Sublink: corresponds to an expression in the query parsing tree. That is, a sublink is a statement in the WHERE or ON clause or in the target list.

In conclusion, a subquery is a range table and a sublink is an expression in the query parse tree. A sublink can be found in constraint conditions and expressions. In GaussDB, sublinks can be classified into the following types:

- exist_sublink: corresponds to the **EXIST** and **NOT EXIST** statements.
- any_sublink: corresponds to the *op* ANY(SELECT...) statement. *op* can be the <, >, or = operator. IN/NOT IN (SELECT...) also belongs to this type.
- all_sublink: corresponds to the *op* ALL(SELECT...) statement. *op* can be the <, >, or = operator.
- rowcompare_sublink: corresponds to the RECORD *op* (SELECT...) statement.

- expr_sublink: corresponds to the (SELECT with a single target list item...) statement.
- array_sublink: corresponds to the ARRAY(SELECT...) statement.
- cte_sublink: corresponds to the WITH(...) query statement.

The exist_sublink and any_sublink are pulled up by the optimization engine of GaussDB. In addition, expr_sublink can also be pulled up. However, because of the flexible use of subqueries in SQL statements, complex subqueries may affect query performance. If you do not want to pull up expr_sublink, set the GUC parameter **rewrite_rule**. Subqueries are classified into non-correlated subqueries and correlated subqueries.

- **Non-correlated subqueries**

The execution of a subquery is independent from attributes of the outer query. In this way, a subquery can be executed before outer queries.

Example:

```
gaussdb=# select t1.c1,t1.c2
from t1
where t1.c1 in (
  select c2
  from t2
  where t2.c2 IN (2,3,4)
);
                                QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Right Semi Join
   Hash Cond: (t2.c2 = t1.c1)
   -> Streaming(type: REDISTRIBUTE)
       Spawn on: All datanodes
       -> Seq Scan on t2
           Filter: (c2 = ANY ('{2,3,4}'::integer[]))
   -> Hash
       -> Seq Scan on t1
(10 rows)
```

- **Correlated subqueries**

The execution of a subquery depends on some attributes (used as AND conditions of the subquery) of outer queries. In the following example, **t1.c1** in the **t2.c1 = t1.c1** condition is a correlated attribute. Such a subquery depends on outer queries and needs to be executed once for each outer query.

Example:

```
gaussdb=# select t1.c1,t1.c2
from t1
where t1.c1 in (
  select c2
  from t2
  where t2.c1 = t1.c1 AND t2.c2 in (2,3,4)
);
                                QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
   -> Result
       Filter: (t2.c1 = t1.c1)
       -> Materialize
           -> Streaming(type: BROADCAST)
```

```

Spawn on: All datanodes
-> Seq Scan on t2
    Filter: (c2 = ANY ('{2,3,4}'::integer[]))
(12 rows)

```

Sublink Optimization on GaussDB

To optimize a sublink, a subquery is pulled up to join with tables in outer queries, preventing the subquery from being converted into a plan involving subplans and broadcast. You can run the EXPLAIN statement to check whether a sublink is converted into the combination of a subplan and broadcast.

Example:

```

gaussdb=# EXPLAIN (COSTS OFF) SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN (SELECT c2 FROM t2
WHERE t2.c1 = t1.c1);
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on t1
    Filter: (SubPlan 1)
    SubPlan 1
-> Result
    Filter: (t2.c1 = t1.c1)
-> Materialize
    -> Streaming(type: BROADCAST)
        Spawn on: All datanodes
        -> Seq Scan on t2
(11 rows)

```

- **Sublink-release scenarios supported by GaussDB**

- Pulling up the IN sublink

- The subquery cannot contain columns in the outer query (columns in more outer queries are allowed).
- The subquery cannot contain volatile functions.

Example:

```

gaussdb=# EXPLAIN (COSTS OFF) SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN (SELECT c2
FROM t2 WHERE t2.c1 = 1);
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Nested Loop Semi Join
    Join Filter: (t1.c1 = t2.c2)
    -> Seq Scan on t1
    -> Materialize
        -> Streaming(type: REDISTRIBUTE)
            Spawn on: datanode1
            -> Seq Scan on t2
                Filter: (c1 = 1)
(10 rows)

```

- Pulling up the EXISTS sublink

The WHERE clause must contain a column in the outer query. Other parts of the subquery cannot contain the column. Other restrictions are as follows:

- The subquery must contain the FROM clause.
- The subquery cannot contain the WITH clause.

- The subquery cannot contain aggregate functions.
- The subquery cannot contain a SET, SORT, LIMIT, WindowAgg, or HAVING operation.
- The subquery cannot contain volatile functions.

Example:

```
gaussdb=# EXPLAIN (COSTS OFF) SELECT t1.c1, t1.c2 FROM t1 WHERE exists (SELECT c2
FROM t2 WHERE t2.c1 = t1.c1);
QUERY PLAN
```

```
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Semi Join
    Hash Cond: (t1.c1 = t2.c1)
-> Seq Scan on t1
-> Hash
    -> Seq Scan on t2
```

(7 rows)

- Pulling up an equivalent correlated query containing aggregate functions

The WHERE condition of the subquery must contain a column from the outer query. Equivalence comparison must be performed between this column and related columns in tables of the subquery. These conditions must be connected using AND. Other parts of the subquery cannot contain the column. Other restrictions are as follows:

- The columns in the expression in the WHERE condition of the subquery must exist in tables.
- After the SELECT keyword of the subquery, there must be only one output column. The output column must be an aggregate function (for example, MAX), and the parameter (for example, **t2.c2**) of the aggregate function cannot be columns of a table (for example, **t1**) in outer queries. The aggregate function cannot be COUNT.

For example, the following subquery can be pulled up:

```
select * from t1 where c1 >(
    select max(t2.c1) from t2 where t2.c1=t1.c1
);
```

The following subquery cannot be pulled up because the subquery has no aggregate function:

```
select * from t1 where c1 >(
    select t2.c1 from t2 where t2.c1=t1.c1
);
```

The following subquery cannot be pulled up because the subquery has two output columns:

```
select * from t1 where (c1,c2) >(
    select max(t2.c1),min(t2.c2) from t2 where t2.c1=t1.c1
);
```

- The subquery must be a FROM clause.
- The subquery cannot contain a GROUP BY, HAVING, or SET operation.
- The subquery can only be an inner join.

For example, the following subquery cannot be pulled up:

```
select * from t1 where c1 >(
    select max(t2.c1) from t2 full join t3 on (t2.c2=t3.c2) where t2.c1=t1.c1
);
```

- The target list of the subquery cannot contain the function that returns a set.
- The WHERE condition of the subquery must contain a column from the outer query. Equivalence comparison must be performed between this column and related columns in tables of the subquery. These conditions must be connected using AND. Other parts of the subquery cannot contain the column. For example, the following innermost sublink can be pulled up:

```
select * from t3 where t3.c1=(
    select t1.c1
    from t1 where c1 >(
        select max(t2.c1) from t2 where t2.c1=t1.c1
    ));
```

If another condition is added to the subquery in the previous example, the subquery cannot be pulled up because the innermost subquery references to the column in the outer and outer query. Example:

```
select * from t3 where t3.c1=(
    select t1.c1
    from t1 where c1 >(
        select max(t2.c1) from t2 where t2.c1=t1.c1 and t3.c1>t2.c2
    ));
```

- Pulling up a sublink in the OR clause

If the WHERE condition contains an EXIST correlated sublink connected by OR:

Example:

```
select a, c from t1
where t1.a = (select avg(a) from t3 where t1.b = t3.b) or
exists (select * from t4 where t1.c = t4.c);
```

The process of pulling up such a sublink is as follows:

- i. Extract **opExpr** from the OR clause in the WHERE condition. The value is **t1.a = (select avg(a) from t3 where t1.b = t3.b)**.
- ii. The **opExpr** contains a subquery. If the subquery can be pulled up, the subquery is rewritten as **select avg(a), t3.b from t3 group by t3.b**, generating the NOT NULL condition **t3.b is not null**. The **opExpr** is replaced by this NOT NULL condition. In this case, the SQL statement changes to:

```
select a, c
from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b) as t3 on (t1.a = avg
and t1.b = t3.b)
where t3.b is not null or exists (select * from t4 where t1.c = t4.c);
```

- iii. Extract the EXISTS sublink **exists (select * from t4 where t1.c = t4.c)** from the OR clause to check whether the sublink can be pulled up. If it can be pulled up, it is converted into **select t4.c from t4 group by t4.c**, generating the NOT NULL condition **t4.c is not null**. In this case, the SQL statement changes to:

```
select t1.a, t1.c from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b) as t3 on
(t1.a = avg and t1.b = t3.b) left join (select t5.c from t5 group by t5.c) as t5 on (t1.c =
t5.c) where t3.b is not null or t5.c is not null;
```

- **Sublink-release scenarios not supported by GaussDB**

Except the sublinks described above, all the other sublinks cannot be pulled up. In this case, a join subquery is planned as the combination of subplans and broadcast. As a result, if inner tables have a large amount of data, query performance may be poor.

If a correlated subquery joins with two tables in outer queries, the subquery cannot be pulled up. You need to change the outer query into a WITH clause and then perform the join.

Example:

```
select distinct t1.a, t2.a
from t1 left join t2 on t1.a=t2.a and not exists (select a,b from test1 where test1.a=t1.a and test1.b=t2.a);
```

The outer query is changed into:

```
with temp as
(
    select * from (select t1.a as a, t2.a as b from t1 left join t2 on t1.a=t2.a)
)
select distinct a,b
from temp
where not exists (select a,b from test1 where temp.a=test1.a and temp.b=test1.b);
```

- The subquery (without COUNT) in the target list cannot be pulled up.

Example:

```
gaussdb=# explain (costs off)
select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
from t1
where t1.c2 > 10;
```

The execution plan is as follows:

```
gaussdb=# explain (costs off)
select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
from t1
where t1.c2 > 10;
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on t1
Filter: (c2 > 10)
SubPlan 1
-> Result
Filter: (t1.c1 = t2.c1)
-> Materialize
-> Streaming(type: BROADCAST)
Spawn on: All datanodes
-> Seq Scan on t2
(11 rows)
```

The correlated subquery is displayed in the target list (query return list). Values need to be returned even if the condition **t1.c1=t2.c1** is not met. Therefore, use a right outer join to join **t2** and **t1** so that the SSQ can return padding values when the condition **t1.c1=t2.c1** is not met.

 **NOTE**

ScalarSubQuery (SSQ) and Correlated-ScalarSubQuery (CSSQ) are described as follows:

- SSQ: a sublink that returns a scalar value of a single row with a single column
- CSSQ: an SSQ containing correlation conditions

The preceding SQL statement can be changed into:

```
with ssq as
(
  select * from t1 where t1.c2 > 10
)
select t2.c2,ssq.c2 from t2 right join ssq on ssq.c1 = t2.c1;
```

The execution plan after the change is as follows:

```
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Right Join
  Hash Cond: (t2.c1 = t1.c1)
  -> Seq Scan on t2
  -> Hash
      -> Seq Scan on t1
          Filter: (c2 > 10)
(8 rows)
```

In the preceding example, the SSQ in the target list is pulled up to right join, preventing poor performance caused by the plan involving subplans and broadcast when the table (**t2**) in the subquery is too large.

- The subquery (with COUNT) in the target list cannot be pulled up.

Example:

```
select (select count(*) from t2 where t2.c1=t1.c1) cnt, t1.c1, t3.c1
from t1,t3
where t1.c1=t3.c1 order by cnt, t1.c1;
```

The execution plan is as follows:

```
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Sort
  Sort Key: ((SubPlan 1)), t1.c1
  -> Hash Join
    Hash Cond: (t1.c1 = t3.c1)
    -> Seq Scan on t1
    -> Hash
        -> Seq Scan on t3
  SubPlan 1
  -> Aggregate
    -> Result
      Filter: (t2.c1 = t1.c1)
    -> Materialize
      -> Streaming(type: BROADCAST)
        Spawn on: All datanodes
      -> Seq Scan on t2
(17 rows)
```

The correlated subquery is displayed in the target list (query return list). Values need to be returned even if the condition **t1.c1=t2.c1** is not met. Therefore, use left outer join to join **T1** and **T2** so that SSQ can return padding values when the condition **t1.c1=t2.c1** is not met. However, COUNT is used, which requires that **0** is returned when the condition is not met. Therefore, case-when NULL then 0 else count(*) can be used.

The preceding SQL statement can be changed into:

```
with ssq as
(
  select count(*) cnt, c1 from t2 group by c1
)
select case when
  ssq.cnt is null then 0
  else ssq.cnt
end cnt, t1.c1, t3.c1
```

```
from t1 left join ssq on ssq.c1 = t1.c1,t3
where t1.c1 = t3.c1
order by ssq.cnt, t1.c1;
```

The execution plan after the change is as follows:

```
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Sort
  Sort Key: (count(*)), t1.c1
  -> Hash Join
    Hash Cond: (t1.c1 = t3.c1)
    -> Hash Left Join
      Hash Cond: (t1.c1 = t2.c1)
      -> Seq Scan on t1
      -> Hash
        -> HashAggregate
          Group By Key: t2.c1
          -> Seq Scan on t2
    -> Hash
      -> Seq Scan on t3
(15 rows)
```

- Non-equivalent correlated subqueries cannot be pulled up.

Example:

```
select t1.c1, t1.c2
from t1
where t1.c1 = (select agg() from t2.c2 > t1.c2);
```

Non-equivalent correlated subqueries cannot be pulled up. You can perform join twice (one CorrelationKey and one rownum self-join) to rewrite the statement.

You can rewrite the statement in either of the following ways:

- Subquery rewriting


```
select t1.c1, t1.c2
from t1, (
  select t1.rowid, agg() aggref
  from t1,t2
  where t1.c2 > t2.c2 group by t1.rowid
) dt /* derived table */
where t1.rowid = dt.rowid AND t1.c1 = dt.aggref;
```

- CTE rewriting


```
WITH dt as
(
  select t1.rowid, agg() aggref
  from t1,t2
  where t1.c2 > t2.c2 group by t1.rowid
)
select t1.c1, t1.c2
from t1, dt
where t1.rowid = dt.rowid AND
t1.c1 = dt.aggref;
```

NOTICE

- Currently, GaussDB does not have an effective way to provide globally unique row IDs for tables and intermediate result sets. Therefore, the rewriting is difficult. It is recommended that this issue is avoided at the service layer or by using **t1.xc_nodeid + t1.ctid** to associate row IDs. However, the high repetition rate of **xc_nodeid** leads to low association efficiency, and **xc_node_id+ctid** cannot be used as the join condition of a hash join.
- If the AGG type is COUNT(*), **0** is used for data padding when CASE-WHEN is not matched. If the type is not COUNT(*), **NULL** is used.
- CTE rewriting works better by using sharescan.

More Optimization Examples

Example 1: Change the base table to a replication table and create an index on the filter column.

```
create table master_table (a int);
create table sub_table(a int, b int);
select a from master_table group by a having a in (select a from sub_table);
```

In this example, a correlated subquery is contained. To improve the query performance, you can change **sub_table** to a replication table when creating a table and create an index on the **a** column.

Example 2: Modify the SELECT statement to change the subquery to a join relationship between the main table and the parent query, or modify the subquery to improve the query performance. Ensure that the subquery to be used is semantically correct.

```
gaussdb=# explain (costs off)select * from master_table as t1 where t1.a in (select t2.a from sub_table as t2 where t1.a = t2.b);
```

QUERY PLAN

```
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on master_table t1
   Filter: (SubPlan 1)
   SubPlan 1
   -> Result
       Filter: (t1.a = t2.b)
       -> Materialize
           -> Streaming(type: BROADCAST)
               Spawn on: All datanodes
           -> Seq Scan on sub_table t2
```

(11 rows)

In the preceding example, a subplan is used. To remove the subplan, you can modify the statement as follows:

```
gaussdb=# explain(costs off) select * from master_table as t1 where exists (select t2.a from sub_table as t2 where t1.a = t2.b and t1.a = t2.a);
```

QUERY PLAN

```
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Semi Join
   Hash Cond: (t1.a = t2.b)
   -> Seq Scan on master_table t1
```

```
-> Hash
  -> Streaming(type: REDISTRIBUTE)
      Spawn on: All datanodes
  -> Seq Scan on sub_table t2
(9 rows)
```

In this way, the subplan is replaced by the semi-join between the two tables, greatly improving the execution efficiency.

6.7.4 Optimizing Statistics

Context

GaussDB generates optimal execution plans based on the cost estimation. Optimizers need to estimate the number of data rows and the cost based on statistics collected using ANALYZE. Therefore, the statistics is vital for the estimation of the number of rows and cost. Global statistics are collected using ANALYZE: **relpages** and **reltuples** in the **pg_class** table; **stadistinct**, **stanullfrac**, **stanumbersN**, **stavaluesN**, and **histogram_bounds** in the **pg_statistic** table.

Example 1: Poor Query Performance Due to the Lack of Statistics

In most cases, the lack of statistics about tables or columns involved in the query greatly affects the query performance.

The table structure is as follows:

```
CREATE TABLE LINEITEM
(
L_ORDERKEY      BIGINT      NOT NULL
, L_PARTKEY      BIGINT      NOT NULL
, L_SUPPKEY      BIGINT      NOT NULL
, L_LINENUMBER   BIGINT      NOT NULL
, L_QUANTITY     DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL
, L_DISCOUNT    DECIMAL(15,2) NOT NULL
, L_TAX          DECIMAL(15,2) NOT NULL
, L_RETURNFLAG   CHAR(1)     NOT NULL
, L_LINESTATUS   CHAR(1)     NOT NULL
, L_SHIPDATE     DATE        NOT NULL
, L_COMMITDATE   DATE        NOT NULL
, L_RECEIPTDATE  DATE        NOT NULL
, L_SHIPINSTRUCT CHAR(25)    NOT NULL
, L_SHIPMODE     CHAR(10)    NOT NULL
, L_COMMENT      VARCHAR(44) NOT NULL
) distribute by hash(L_ORDERKEY);

CREATE TABLE ORDERS
(
O_ORDERKEY      BIGINT      NOT NULL
, O_CUSTKEY      BIGINT      NOT NULL
, O_ORDERSTATUS  CHAR(1)     NOT NULL
, O_TOTALPRICE   DECIMAL(15,2) NOT NULL
, O_ORDERDATE    DATE        NOT NULL
, O_ORDERPRIORITY CHAR(15)   NOT NULL
, O_CLERK        CHAR(15)    NOT NULL
, O_SHIPPRIORITY BIGINT      NOT NULL
, O_COMMENT      VARCHAR(79) NOT NULL
) distribute by hash(O_ORDERKEY);
```

The query statements are as follows:

```
explain verbose select
count(*) as numwait
```

```
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

If such an issue occurs, you can use the following methods to check whether statistics in tables or columns has been collected using ANALYZE.

1. Execute EXPLAIN VERBOSE to analyze the execution plan and check the warning information:

```
WARNING:Statistics in some tables or columns(public.lineitem.l_receiptdate,
public.lineitem.l_commitdate, public.lineitem.l_orderkey, public.lineitem.l_suppkey,
public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
HINT:Do analyze for them in order to generate optimized plan.
```

2. Check whether the following information exists in the log file in the **pg_log** directory. If it does, the poor query performance was caused by the lack of statistics in some tables or columns.

```
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables
or columns(public.lineitem.l_receiptdate, public.lineitem.l_commitdate, public.lineitem.l_orderkey,
public.linei
tem.l_suppkey, public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] HINT:Do analyze for them in
order to generate optimized plan.
```

By using any of the preceding methods, you can identify tables or columns whose statistics have not been collected using ANALYZE. You can execute ANALYZE to warnings or tables and columns recorded in logs to resolve the problem.

Example 2: Optimization is Not Accurate When Intermediate Results Exist in the Query Where JOIN Is Used for Multiple Tables

Symptom: Query the personnel who have registered in an Internet cafe within 15 minutes before and after the registration of a specified person.

```
SELECT
C.WBM,
C.DZQH,
C.DZ,
B.ZJHM,
B.SWKSSJ,
B.XWSJ
FROM
b_zyk_wbswxx A,
b_zyk_wbswxx B,
b_zyk_wbcs C
WHERE
A.ZJHM = '522522*****3824'
AND A.WBDM = B.WBDM
AND A.WBDM = C.WBDM
AND abs(to_date(A.SWKSSJ,'yyyymmddHH24MISS') - to_date(B.SWKSSJ,'yyyymmddHH24MISS')) <
```

```
INTERVAL '15 MINUTES'
ORDER BY
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

Figure 6-4 shows the execution plan. This query takes about 12s.

Figure 6-4 Using an unlogged table (1)

```
QUERY PLAN
-----
Limit (cost=221021.41..221021.43 rows=10 width=120)
-> Sort (cost=221021.41..221022.01 rows=240 width=120)
    Sort Key: b.swkssj, b.zjhm
    -> Streaming (type: GATHER) (cost=221015.62..221016.22 rows=240 width=120)
        Node/s: All datanodes
        -> Limit (cost=3209.98..3209.01 rows=10 width=120)
            -> Sort (cost=3209.98..3211.60 rows=1048 width=120)
                Sort Key: b.swkssj, b.zjhm
                -> Nested Loop (cost=23.27..9186.34 rows=1048 width=120)
                    Join Filter: (((a.zjhm)::text <> (b.zjhm)::text) AND ((a.wbdm)::text = (b.wbdm)::text)
                    AND (abs(((to_date((a.swkssj)::text, 'yyyymmddHH24MISS')::text
                    - to_date((b.swkssj)::text, 'yyyymmddHH24MISS')::text))::numeric) < .010416666666667))
                    -> Streaming (type: BROADCAST) (cost=0.00..6.33 rows=24 width=135)
                        Spawn on: All datanodes
                        -> Nested Loop (cost=0.00..106.80 rows=1 width=135)
                            -> Streaming (type: BROADCAST) (cost=0.00..24.75 rows=264 width=48)
                                Spawn on: All datanodes
                                -> Partition Iterator (cost=0.00..48.44 rows=11 width=48)
                                    Iterations: 25
                                    -> Partitioned Index Scan using idx_b_zyk_wbvwxx_zjhm on b_zyk_wbvwxx a (cost=0.00..48.44 rows=11 width=48)
                                        Index Cond: ((zjhm)::text = '522522*****3824')::text
                                        Selected Partitions: 1..25
                                    -> Index Scan using idx_b_zyk_wbcs_wbdm on b_zyk_wbcs c (cost=0.00..2.82 rows=1 width=87)
                                        Index Cond: ((wbdm)::text = (a.wbdm)::text)
                            -> Partition Iterator (cost=23.27..7306.33 rows=2454 width=63)
                                Iterations: 25
                                -> Partitioned Bitmap Heap Scan on b_zyk_wbvwxx b (cost=23.27..7306.33 rows=2454 width=63)
                                    Recheck Cond: ((wbdm)::text = (c.wbdm)::text)
                                    Filter: ('522522198405243824')::text <> (zjhm)::text
                                    Selected Partitions: 1..25
                                -> Partitioned Bitmap Index Scan on idx_b_zyk_wbvwxx_wbdm (cost=0.00..22.65 rows=2454 width=0)
                                    Index Cond: ((wbdm)::text = (c.wbdm)::text)
```

Optimization analysis:

1. In the execution plan, index scan is used for node scanning, the **Join Filter** calculation in the external **NEST LOOP JOIN** statement consumes most of the query time, and the calculation uses the string addition and subtraction, and unequal-value comparison.
2. Use an unlogged table to record the Internet access time of the specified person. The start time and end time are processed during data insertion, and this reduces subsequent addition and subtraction operations.

```
// Create a temporary unlogged table.
CREATE UNLOGGED TABLE temp_tsw
(
ZJHM NVARCHAR2(18),
WBDM NVARCHAR2(14),
SWKSSJ_START NVARCHAR2(14),
SWKSSJ_END NVARCHAR2(14),
WBM NVARCHAR2(70),
DZQH NVARCHAR2(6),
DZ NVARCHAR2(70),
IPDZ NVARCHAR2(39)
)
;
// Insert the Internet access record of the specified person, and process the start time and end time.
INSERT INTO
temp_tsw
SELECT
A.ZJHM,
A.WBDM,
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') - INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') + INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
```

```

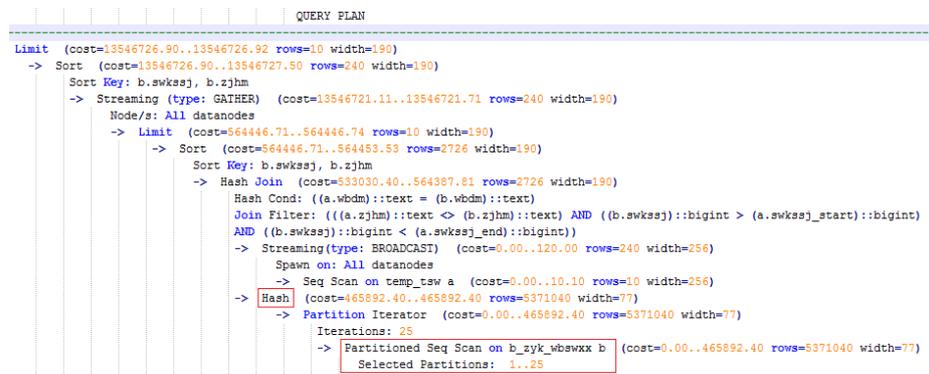
B.WBM,B.DZQH,B.DZ,B.IPDZ
FROM
b_zyk_wbswxx A,
b_zyk_wbcs B
WHERE
A.ZJHM='522522*****3824' AND A.WBDM = B.WBDM
;

// Query the personnel who have registered in an Internet cafe before and after 15 minutes of the
registration of the specified person. Convert their ID card number format to int8 in comparison.
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
order by
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;

```

The query takes about 7s. [Figure 6-5](#) shows the execution plan.

Figure 6-5 Using an unlogged table (2)



3. In the previous plan, a hash join has been executed, and a Hash table has been created for the large table **b_zyk_wbswxx** (Internet access information in the Internet cafe). The table contains large amounts of data, so the creation takes long time.

temp_tsw (Internet access user information) contains only hundreds of records, and an equal-value connection is created between **temp_tsw** and **b_zyk_wbswxx** (Internet access information in the Internet cafe) using **wbdm** (an Internet cafe code). Therefore, if **JOIN** is changed to **NEST LOOP JOIN**, index scan can be used for node scanning, and the performance will be boosted.

4. Execute the following statement to change **JOIN** to **NEST LOOP JOIN**.
SET enable_hashjoin = off;

[Figure 6-6](#) shows the execution plan. The query takes about 3s.

Figure 6-6 Using an unlogged table (3)

```

QUERY PLAN
-----
Limit (cost=240002336196.14..240002336196.17 rows=10 width=190)
-> Sort (cost=240002336196.14..240002336196.74 rows=240 width=190)
    Sort Key: b.swkssj, b.zjhm
    -> Streaming (type: GATHER) (cost=240002336190.35..240002336190.95 rows=240 width=190)
        Node/s: All datanodes
        -> Limit (cost=10000097341.26..10000097341.29 rows=10 width=190)
            Sort (cost=10000097341.26..10000097348.08 rows=2726 width=190)
                Sort Key: b.swkssj, b.zjhm
                -> Nested Loop (cost=10000000000.00..10000097282.36 rows=2726 width=190)
                    -> Streaming (type: BROADCAST) (cost=0.00..120.00 rows=240 width=256)
                        Spawn on: All datanodes
                        -> Seq Scan on temp_tsw a (cost=0.00..10.10 rows=10 width=256)
                    -> Partition Iterator (cost=0.00..9648.34 rows=273 width=77)
                        Iterations: 25
                        -> Partitioned Index Scan using idx_b_zyk_wbswxx_wbdm on b_zyk_wbswxx b (cost=0.00..9648.34 rows=273 width=77)
                            Index Cond: ((wbdm)::text = (a.wbdm)::text)
                            Filter: (((a.zjhm)::text <> (zjhm)::text) AND ((swkssj)::bigint > (a.swkssj_start)::bigint)
                                AND ((swkssj)::bigint < (a.swkssj_end)::bigint))
                            Selected Partitions: 1..25
(10 rows)

```

5. Save the query result set in the unlogged table for paging display.

If paging display needs to be achieved on the upper-layer application page, change the **offset** value to determine the result set on the target page. In this way, the previous query statement will be executed every time after a page turning operation, which causes long response latency.

To resolve this problem, the unlogged table is recommended to save the result set.

```

// Create an unlogged table to save the result set.
CREATE UNLOGGED TABLE temp_result
(
WBM    NVARCHAR2(70),
DZQH   NVARCHAR2(6),
DZ     NVARCHAR2(70),
IPDZ   NVARCHAR2(39),
ZJHM   NVARCHAR2(18),
XM     NVARCHAR2(30),
SWKSSJ date,
XWSJ   date,
SWZDH  NVARCHAR2(32)
);

// Insert the result set to the unlogged table. The insertion takes about 3s.
INSERT INTO
temp_result
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
;

// Perform paging query on the result set. The paging query takes about 10 ms.
SELECT
*
FROM
temp_result
ORDER BY
SWKSSJ,

```

```
ZJHM
LIMIT 10 OFFSET 0;
```

CAUTION

Collecting more accurate statistics usually improves the query performance, but may also deteriorate the performance. If the performance deteriorates, you can:

- Restore to the default statistics.
- Use hints to force the optimizer to use the optimal query plan. (For details, see [Hint-based Tuning](#).)

6.7.5 Optimizing Operators

Context

A query statement needs to go through multiple operator procedures to generate the final result. Sometimes, the overall query performance deteriorates due to long execution time of certain operators, which are regarded as bottleneck operators. In this case, you need to execute the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** command to view the bottleneck operators, and then perform optimization.

For example, in the following execution process, the execution time of the Hashagg operator accounts for about 66% [(51016-13535)/56476 ≈ 66%] of the total execution time. Therefore, the Hashagg operator is the bottleneck operator for this query. Optimize this operator first.

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Row Adapter	56476.397	10000000	237060	19KB			20	2093222.75
2	-> Vector Streaming (type: GATHER)	85664.220	10000000	237060	249KB			20	2093222.75
3	-> Vector Hash Aggregate	[55124.685,55132.180]	10000000	237060	[259489KB, 29441KB]	16MB	[20,20]	20	2093406.50
4	-> Vector Streaming (type: REDISTRIBUTE)	[52519.781,53709.779]	339364604	4856184	[12199KB, 12199KB]	1MB		20	10461210.85
5	-> Vector Hash Aggregate	[35875.636,51016.424]	339364604	4856184	[722850KB, 746894KB]	16MB	[20,20]	20	10467195.65
6	-> Vector Partition Traversal	[9035.202,13565.894]	97000000	935838997	[9KB, 9KB]	1MB		20	10195891.68
7	-> Partitioned CStore Scan on xuzi_e_up_day_energy_mv_1	[9015.645,13535.346]	97000000	935838997	[845KB, 845KB]	1MB		20	10195891.68

Example

Example 1: Scan the base table. For queries requiring large volume of data filtering, such as point queries or queries that need range scanning, a full table scan using SeqScan will take a long time. To facilitate scanning, you can create indexes on the condition column and select IndexScan for index scanning.

```
gaussdb=# explain (analyze on,costs off) select * from t1 where c2=10004;
id | operation | A-time | A-rows | Peak Memory | A-width
-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 20.040 | 5 | 85KB |
2 | -> Seq Scan on t1 | [17.239,17.376] | 5 | [18KB,18KB] |
(2 rows)

Predicate Information (identified by plan id)
-----
2 --Seq Scan on t1
Filter: (c2 = 10004)
Rows Removed by Filter: 90002
(3 rows)
gaussdb=# create index idx on t1(c2);
CREATE INDEX
gaussdb=# explain (analyze on,costs off) select * from t1 where c2=10004;
```

id	operation	A-time	A-rows	Peak Memory	A-width
1	-> Streaming (type: GATHER)	3.206	5	85KB	
2	-> Index Scan using idx on t1	[0.122,0.146]	5	[73KB,73KB]	

(2 rows)

Predicate Information (identified by plan id)

2	--Index Scan using idx on t1	Index Cond: (c2 = 10004)			
---	------------------------------	--------------------------	--	--	--

(2 rows)

In this example, the full table scan filters large amounts of data and returns 5 records. After an index has been created on the **c2** column, the scanning efficiency is significantly boosted and the duration of IndexScan is reduced from 20 ms to 3 ms.

Example 2: If NestLoop is used for joining tables with a large number of rows, the join may take a long time. In the following example, NestLoop takes 5s. If **enable_mergejoin** is set to **off** to disable merge join and **enable_nestloop** is set to **off** to disable NestLoop so that the optimizer selects hash join, the join duration is reduced to 86 ms.

```
gaussdb=# explain analyze select count(*) from t2,t1 where t1.c1=t2.c2;
```

id	operation	A-time	A-rows	E-rows	Peak Memory	A-width	E-width	E-costs
1	-> Aggregate	5070.296	1	1	14KB		8	
2	-> Streaming (type: GATHER)	5070.219	2	2	81KB		8	
3	-> Aggregate	[4828.705,5062.289]	2	2	[11KB,11KB]		8	
4	-> Nested Loop (5,6)	[4828.565,5062.142]	996	40	[4KB,4KB]			
5	-> Seq Scan on t1	[13.574,14.508]	90007	20000	[15KB,15KB]			
6	-> Materialize	[1508.956,1579.488]	22413670	20	[35KB,36KB]			
7	-> Streaming(type: REDISTRIBUTE)	[55.825,56.842]	498	20	[44KB,44KB]			
8	-> Seq Scan on t2	[0.105,0.132]	498	20	[13KB,13KB]		4	

(8 rows)

Predicate Information (identified by plan id)

4	--Nested Loop (5,6)	Join Filter: (t2.c2 = t1.c1)	Rows Removed by Join Filter: 22412672					
---	---------------------	------------------------------	---------------------------------------	--	--	--	--	--

(3 rows)

After the parameters are set:

```
gaussdb=# set enable_mergejoin=off;
SET
gaussdb=# set enable_nestloop=off;
SET
gaussdb=# explain analyze select count(*) from t2,t1 where t1.c1=t2.c2;
```

id	operation	A-time	A-rows	E-rows	Peak Memory	A-width	E-width	E-costs
1	-> Aggregate	92.911	1	1	14KB		8	224.45
2	-> Streaming (type: GATHER)	92.855	2	2	81KB		8	
3	-> Aggregate	[84.295,87.102]	2	2	[11KB,11KB]		8	

```

224.36
 4 |      -> Hash Join (5,6)          | [84.171,86.966] | 996 | 40 | [6KB,6KB] | | 0 |
224.30
 5 |      -> Seq Scan on t1          | [11.885,13.103] | 90007 | 20000 | [15KB,15KB] | | 4 |
184.00
 6 |      -> Hash                    | [55.895,56.072] | 498 | 21 | [292KB,292KB] | [20,20] | 4 |
14.31
 7 |      -> Streaming(type: REDISTRIBUTE) | [55.601,55.771] | 498 | 20 | [44KB,44KB] | | |
| 4 | 14.31
 8 |      -> Seq Scan on t2          | [0.118,0.143] | 498 | 20 | [13KB,13KB] | | 4 |
13.13
(8 rows)

Predicate Information (identified by plan id)
-----
 4 --Hash Join (5,6)
      Hash Cond: (t1.c1 = t2.c2)
(2 rows)

```

Example 3: Generally, query performance can be improved by selecting **HashAgg**. If **Sort** and **GroupAgg** are used for a large result set, you need to set **enable_sort** to **off**. **HashAgg** consumes less time than **Sort** and **GroupAgg**.

```

gaussdb=# explain analyze select count(*) from t1 group by c2;
 id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
 1 | -> GroupAggregate | 244.817 | 40000 | 5000 | 15KB | | | 12 |
2131.52
 2 | -> Sort | 156.344 | 40000 | 10000 | 5603KB | | | 12 |
2131.52
 3 | -> Streaming (type: GATHER) | 91.595 | 40000 | 10000 | 82KB | | | |
12 | 1442.14
 4 | -> GroupAggregate | [90.317,96.852] | 40000 | 10000 | [12KB,12KB] | 16MB | | |
| 12 | 973.39
 5 | -> Sort | [59.775,64.724] | 90007 | 20000 | [5MB,5MB] | 16MB | | |
[896220,903920] | 4 | 873.39
 6 | -> Seq Scan on t1 | [18.092,21.033] | 90007 | 20000 | [12KB,12KB] | 1MB | | |
| 4 | 184.00
(6 rows)

```

After the parameters are set:

```

gaussdb=# set enable_sort=off;
SET
gaussdb=# explain analyze select count(*) from t1 group by c2;
 id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
 1 | -> HashAggregate | 228.260 | 40000 | 5000 | 6663KB | | | 12 | 752.75
 2 | -> Streaming (type: GATHER) | 95.506 | 40000 | 10000 | 82KB | | | | 12 |
752.75
 3 | -> HashAggregate | [63.974,71.290] | 40000 | 10000 | [3MB,3MB] | 16MB | [20,20] | |
12 | 284.00
 4 | -> Seq Scan on t1 | [17.578,21.204] | 90007 | 20000 | [12KB,12KB] | 1MB | | | 4 |
184.00
(4 rows)

```

6.8 Experience in Rewriting SQL Statements

Based on the SQL execution mechanism and a large number of practices, SQL statements can be optimized by following certain rules to enable the database to execute SQL statements more quickly and obtain correct results. You can comply with these rules to improve service query efficiency.

- Replace **UNION** with **UNION ALL**.
UNION eliminates duplicate rows while merging two result sets but **UNION ALL** merges the two result sets without deduplication. Therefore, replace **UNION** with **UNION ALL** if you are sure that the two result sets do not contain duplicate rows based on the service logic.
- Add **NOT NULL** to the join columns.
If there are many NULL values in the **JOIN** columns, you can add the filter criterion **IS NOT NULL** to filter data in advance to improve the **JOIN** efficiency.
- Convert **NOT IN** to **NOT EXISTS**.
nestloop anti join must be used to implement **NOT IN**, and **hash anti join** is required for **NOT EXISTS**. If no NULL value exists in the **JOIN** columns, **NOT IN** is equivalent to **NOT EXISTS**. Therefore, if you are sure that no NULL value exists, you can convert **NOT IN** to **NOT EXISTS** to generate **hash join** and to improve the query performance.

As shown in the following statement, the **t2.d2** column does not contain null values (it is set to **NOT NULL**) and **NOT EXISTS** is used for the query.

```
SELECT * FROM t1 WHERE NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
```

The generated execution plan is as follows:

Figure 6-7 NOT EXISTS execution plan

```
id | operation
-----+-----
 1 | -> Streaming (type: GATHER)
 2 | -> Hash Anti Join (3, 4)
 3 | -> Seq Scan on t1
 4 | -> Hash
 5 | -> Streaming(type: REDISTRIBUTE)
 6 | -> Seq Scan on t2
(6 rows)

Predicate Information (identified by plan id)
-----+-----
 2 --Hash Anti Join (3, 4)
    Hash Cond: (t1.c1 = t2.d2)
(2 rows)
```

- Use **hashagg**.
If a plan involving groupAgg and SORT operations generated by the **GROUP BY** statement is poor in performance, you can set **work_mem** to a larger value to generate a **hashagg** plan, which does not require sorting and improves the performance.
- Replace functions with **CASE** statements.
The GaussDB performance greatly deteriorates if a large number of functions are called. In this case, you can modify the pushdown functions to **CASE** statements.
- Do not use functions or expressions for indexes.
Using functions or expressions for indexes stops indexing. Instead, it enables scanning on the full table.

- Do not use the !=, <, or > operator, NULL, OR, or implicit parameter conversion in WHERE clauses.
- If the values of >= and <= are the same in WHERE condition, change the condition to = because range equivalence class derivation is not supported currently.

For example, change **SELECT * FROM t1 WHERE c1 >= 1 AND c1 <= 1** to **SELECT * FROM t1 WHERE c1 = 1**.

For range queries, the optimizer has a larger error when calculating selectivity than equivalent queries. Therefore, change range queries to equivalent queries as much as possible.

- Split complex SQL statements.

You can split an SQL statement into several ones and save the execution result to a temporary table if the SQL statement is too complex to be tuned using the solutions above, including but not limited to the following scenarios:

- The same subquery is involved in multiple SQL statements of a job and the subquery contains large amounts of data.
- Incorrect plan cost causes a small hash bucket of subquery. For example, the actual number of rows is 10 million, but only 1000 rows are in hash bucket.
- Functions such as **substr** and **to_number** cause incorrect measures for subqueries containing large amounts of data.
- **BROADCAST** subqueries are performed on large tables in multi-DN environment.

6.9 Configuring Key Parameters for SQL Tuning

This section describes key CN configuration parameters that affect GaussDB SQL performance optimization. Contact an administrator to configure parameters.

Table 6-4 CN parameters

Parameter/ Reference Value	Description
enable_nestloop=on	<p>Specifies how the optimizer uses Nest Loop Join. If this parameter is set to on, the optimizer preferentially uses Nest Loop Join. If it is set to off, the optimizer preferentially uses other methods, if any.</p> <p>NOTE If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), execute the following SQL statement: SET enable_nestloop to off;</p> <p>You can determine whether to disable this function based on the actual requirements. Generally, among the three join modes (Nested Loop, Merge Join, and Hash Join), Nested Loop is applicable to scenarios with small data volume or indexes, and Hash Join is applicable to big data analysis scenarios.</p>

Parameter/ Reference Value	Description
enable_bitmapscan=on	<p>Specifies whether the optimizer uses bitmap scan. If the value is on, bitmap scan is used. If the value is off, it is not used.</p> <p>NOTE If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), execute the following SQL statement: SET enable_bitmapscan to off;</p> <p>The bitmap scan applies only in the query condition where a > 1 and b > 1 and indexes are created on columns a and b. However, the performance of bitmapscan is sometimes inferior to that of indexscan. During tuning, if the query performance is poor and bitmapscan operators are in the execution plan, set this parameter to off and check whether the performance is improved.</p>
enable_fast_query_shipping=on	<p>Specifies whether the optimizer uses a distribution framework to execute quick execution plans. If the value is on, the execution plan is generated on both CNs and DN. If the value is off, the distribution framework is used, that is, the execution plan is generated on the CN and then sent to the DN for execution.</p> <p>NOTE If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), execute the following SQL statement: SET enable_fast_query_shipping to off;</p>
enable_hashagg=on	Specifies whether the optimizer uses hash aggregate plans.
enable_hashjoin=on	Specifies whether the optimizer uses hash join plans.
enable_mergejoin=on	Specifies whether the optimizer uses merge join plans.
enable_indexscan=on	Specifies whether the optimizer uses index scan plans.
enable_indexonlyscan=on	Specifies whether the optimizer uses index-only scan plans.
enable_gsiscan=off	Specifies whether the optimizer uses GSI scan plans.
enable_seqscan=on	Specifies whether the optimizer uses sequential scan plans. It is impossible to suppress sequential scans entirely, but setting this variable to off encourages the optimizer to choose other methods if available.

Parameter/ Reference Value	Description
enable_sort=on	Specifies the optimizer sorting order. It is impossible to fully suppress explicit sorting, but setting this variable to off encourages the optimizer to choose other methods if available.
enable_broadcast=on	Specifies whether the optimizer uses data broadcast. In data broadcast, a large amount of data is transferred on the network. When the number of transmission nodes (stream) is large and the estimation is inaccurate, set this parameter to off and check whether the performance is improved.
rewrite_rule	Specifies whether the optimizer enables the LAZYAGG, MAGICSET, PARTIALPUSH, DISABLEREP, UNIQUECHECK, INTARGETLIST, PREDPUSH, PREDPUSHFORCE, PREDPUSHNORMAL, DISABLE_PULLUP_EXPR_SUBLINK, SUBLINK_PULLUP_ENHANCED, DISABLE_PULLUP_NOT_IN_SUBLINK, DISABLE_ROWNUM_PUSHDOWN, or DISABLE_WINDOWAGG_PUSHDOWN rewriting rule.
sql_beta_feature	Specifies whether the optimizer enables the SEL_SEMI_POISSON, NO_UNIQUE_INDEX_FIRST, JOIN_SEL_WITH_CAST_FUNC, SEL_EXPR_INSTR, PARAM_PATH_GEN, RAND_COST_OPT, PARAM_PATH_OPT, PAGE_EST_OPT, CANONICAL_PATHKEY, INDEX_COST_WITH_INDEX_COST_WITH_LEAF_PAGES_ONLY, PREDPUSH_SAME_LEVEL, PARTITION_FDW_ON, DISABLE_BITMAP_COST_WITH_LOSSY_PAGES, or ENABLE_UPSERT_EXECUTE_GPLAN beta feature.
enable_inner_unique_opt	Specifies whether the optimizer uses Inner Unique.

6.10 Hint-based Tuning

6.10.1 Plan Hint Optimization

In plan hints, you can specify a join order; join, stream, and scan operations, the number of rows in a result, and redistribution skew information to tune an execution plan, improving query performance.

GaussDB also provides the SQL patch function. You can create an SQL patch to make hints take effect without modifying service statements.

Description

Plan hints are specified in the following format after keywords such as SELECT, INSERT, UPDATE, DELETE, and MERGE:

```
/*+ <plan hint>*/
```

You can specify multiple hints for a query plan and separate them with spaces. A hint specified for a query plan does not apply to its subquery plans. To specify a hint for a subquery, add the hint following the **SELECT** of this subquery.

Example:

```
select /*+ <plan_hint1> <plan_hint2> */ * from t1, (select /*+ <plan_hint3> */ * from t2) where 1=1;
```

In the preceding command, *<plan_hint1>* and *<plan_hint2>* are the hints of a query, and *<plan_hint3>* is the hint of its subquery.

You can use the EXPLAIN syntax to analyze the plan hint optimization effect. You can use EXPLAIN to view the plan of the target SQL statement after the plan hint is used and check whether the plan meets the requirements to verify the plan hint effect. EXPLAIN has multiple plan display modes, which are controlled by **explain_perf_mode**. In some examples in this section, **explain_perf_mode** is set to **pretty** to display complete plan information. In some examples, **explain_perf_mode** is set to **normal** to simplify the output information.

NOTICE

If a hint is specified in the **CREATE VIEW** statement, the hint will be applied each time this view is used.

If the random plan function is enabled (**plan_mode_seed** is set to a value other than 0), the specified hint will not be used.

Scope

Currently, the following hints are supported:

- Join order hints (**leading**).
- Join operation hints, excluding the **semi join**, **anti join**, and **unique plan** hints.
- Rows hints.
- Stream operation hints.
- Scan operation hints, supporting only the **tablescan**, **indexscan**, **indexonlyscan**, and **gsi** hints.
- Sublink name hints.
- Skew hints, supporting only the skew in the redistribution involving Join or HashAgg.
- Hints of the GUC parameter that takes effect in a query. The hints do not take effect if they are used in views.
- Hints that use the custom plan or generic plan and are valid only for query statements executed by PBE.
- Hints specifying not to expand subqueries.
- Hints specifying that the current query statement does not enter the global plan cache. The hints are valid only when **enable_global_plancache** is enabled and the current statement is executed by PBE.

- Hints for internal table materialization.
- **Bitmapscan** hints
- Hints of the Agg method

Precautions

- Hints do not support **Sort**, **Setop**, or **Subplan**.
- Hints do not support SMP or Node Groups.

Examples

Create tables and indexes.

```
create table t1(c1 int, c2 int, c3 int);
create table t2(c1 int, c2 int, c3 int);
create table t3(c1 int, c2 int, c3 int);
create index it1 on t1(c1,c2);
create index it2 on t2(c1,c2);
create index it3 on t1(c3,c2);
-- The following TPC-H data table needs to be inserted with 10 times of the data to match the provided
plan example.
create table store
(
  s_store_sk          integer          not null,
  s_store_id         char(16)         not null,
  s_rec_start_date   date              ,
  s_rec_end_date     date              ,
  s_closed_date_sk   integer          ,
  s_store_name       varchar(50)      ,
  s_number_employees integer          ,
  s_floor_space      integer          ,
  s_hours            char(20)         ,
  s_manager          varchar(40)      ,
  s_market_id        integer          ,
  s_geography_class  varchar(100)     ,
  s_market_desc      varchar(100)     ,
  s_market_manager   varchar(40)     ,
  s_division_id      integer          ,
  s_division_name    varchar(50)      ,
  s_company_id       integer          ,
  s_company_name     varchar(50)      ,
  s_street_number    varchar(10)      ,
  s_street_name      varchar(60)      ,
  s_street_type      char(15)         ,
  s_suite_number     char(10)         ,
  s_city             varchar(60)      ,
  s_county           varchar(30)      ,
  s_state            char(2)          ,
  s_zip              char(10)         ,
  s_country           varchar(20)     ,
  s_gmt_offset       decimal(5,2)     ,
  s_tax_precentage   decimal(5,2)     ,
  primary key (s_store_sk)
);
create table store_sales
(
  ss_sold_date_sk    integer          ,
  ss_sold_time_sk    integer          ,
  ss_item_sk         integer          not null,
  ss_customer_sk     integer          ,
  ss_cdemo_sk        integer          ,
  ss_hdemo_sk        integer          ,
  ss_addr_sk         integer          ,
  ss_store_sk        integer          ,
  ss_promo_sk        integer          ,
```

```
ss_ticket_number      integer      not null,
ss_quantity           integer
ss_wholesale_cost     decimal(7,2)
ss_list_price         decimal(7,2)
ss_sales_price        decimal(7,2)
ss_ext_discount_amt   decimal(7,2)
ss_ext_sales_price    decimal(7,2)
ss_ext_wholesale_cost decimal(7,2)
ss_ext_list_price     decimal(7,2)
ss_ext_tax            decimal(7,2)
ss_coupon_amt         decimal(7,2)
ss_net_paid           decimal(7,2)
ss_net_paid_inc_tax   decimal(7,2)
ss_net_profit         decimal(7,2)
primary key (ss_item_sk, ss_ticket_number)
);
create table store_returns
(
  sr_returned_date_sk integer
  sr_return_time_sk   integer
  sr_item_sk          integer      not null,
  sr_customer_sk      integer
  sr_cdemo_sk         integer
  sr_hdemo_sk         integer
  sr_addr_sk          integer
  sr_store_sk         integer
  sr_reason_sk        integer
  sr_ticket_number    integer      not null,
  sr_return_quantity  integer
  sr_return_amt       decimal(7,2)
  sr_return_tax       decimal(7,2)
  sr_return_amt_inc_tax decimal(7,2)
  sr_fee              decimal(7,2)
  sr_return_ship_cost decimal(7,2)
  sr_refunded_cash    decimal(7,2)
  sr_reversed_charge  decimal(7,2)
  sr_store_credit     decimal(7,2)
  sr_net_loss         decimal(7,2)
  primary key (sr_item_sk, sr_ticket_number)
);
create table customer
(
  c_customer_sk      integer      not null,
  c_customer_id      char(16)     not null,
  c_current_cdemo_sk integer
  c_current_hdemo_sk integer
  c_current_addr_sk  integer
  c_first_shipto_date_sk integer
  c_first_sales_date_sk integer
  c_salutation       char(10)
  c_first_name       char(20)
  c_last_name        char(30)
  c_preferred_cust_flag char(1)
  c_birth_day        integer
  c_birth_month      integer
  c_birth_year       integer
  c_birth_country    varchar(20)
  c_login            char(13)
  c_email_address    char(50)
  c_last_review_date char(10)
  primary key (c_customer_sk)
);
create table promotion
(
  p_promo_sk      integer      not null,
  p_promo_id      char(16)     not null,
  p_start_date_sk integer
  p_end_date_sk   integer
  p_item_sk       integer
```

```

p_cost          decimal(15,2)      ,
p_response_target integer      ,
p_promo_name    char(50)      ,
p_channel_dmail char(1)      ,
p_channel_email char(1)      ,
p_channel_catalog char(1)    ,
p_channel_tv    char(1)      ,
p_channel_radio char(1)      ,
p_channel_press char(1)      ,
p_channel_event char(1)      ,
p_channel_demo  char(1)      ,
p_channel_details varchar(100) ,
p_purpose         char(15)     ,
p_discount_active char(1)    ,
primary key (p_promo_sk)
);
create table customer_address
(
ca_address_sk integer      not null,
ca_address_id char(16)     not null,
ca_street_number char(10)  ,
ca_street_name   varchar(60) ,
ca_street_type   char(15)   ,
ca_suite_number  char(10)   ,
ca_city          varchar(60) ,
ca_county        varchar(30) ,
ca_state         char(2)    ,
ca_zip          char(10)    ,
ca_country       varchar(20) ,
ca_gmt_offset    decimal(5,2) ,
ca_location_type char(20)   ,
primary key (ca_address_sk)
);
create table item
(
i_item_sk integer      not null,
i_item_id char(16)     not null,
i_rec_start_date date      ,
i_rec_end_date   date      ,
i_item_desc      varchar(200) ,
i_current_price  decimal(7,2) ,
i_wholesale_cost decimal(7,2) ,
i_brand_id      integer      ,
i_brand         char(50)     ,
i_class_id      integer      ,
i_class         char(50)     ,
i_category_id   integer      ,
i_category      char(50)     ,
i_manufact_id   integer      ,
i_manufact      char(50)     ,
i_size          char(20)     ,
i_formulation   char(20)     ,
i_color         char(20)     ,
i_units         char(10)     ,
i_container     char(10)     ,
i_manager_id    integer      ,
i_product_name  char(50)     ,
primary key (i_item_sk)
);

```

The following are the statements (used in most examples in this section) and the original plan without hints for comparing the methods supported by Plan Hint:

```

explain
select i_product_name product_name
,i_item_sk item_sk
,s_store_name store_name
,s_zip store_zip
,ad2.ca_street_number c_street_number
,ad2.ca_street_name c_street_name

```

```
,ad2.ca_city c_city
,ad2.ca_zip c_zip
,count(*) cnt
,sum(ss_wholesale_cost) s1
,sum(ss_list_price) s2
,sum(ss_coupon_amt) s3
FROM store_sales
,store_returns
,store
,customer
,promotion
,customer_address ad2
,item
WHERE ss_store_sk = s_store_sk AND
ss_customer_sk = c_customer_sk AND
ss_item_sk = i_item_sk and
ss_item_sk = sr_item_sk and
ss_ticket_number = sr_ticket_number and
c_current_addr_sk = ad2.ca_address_sk and
ss_promo_sk = p_promo_sk and
i_color in ('maroon','burnished','dim','steel','navajo','chocolate') and
i_current_price between 35 and 35 + 10 and
i_current_price between 35 + 1 and 35 + 15
group by i_product_name
,i_item_sk
,s_store_name
,s_zip
,ad2.ca_street_number
,ad2.ca_street_name
,ad2.ca_city
,ad2.ca_zip
;
```

```
HashAggregate (cost=53.53..53.76 rows=1 width=880)
  Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number,
ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
  -> Streaming (type: GATHER) (cost=53.53..53.76 rows=2 width=880)
    Node/s: All datanodes
    -> HashAggregate (cost=53.10..53.11 rows=2 width=880)
      Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip,
ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
      -> Nested Loop (cost=0.00..53.07 rows=2 width=776)
        -> Streaming (type: REDISTRIBUTE) (cost=0.00..46.36 rows=2 width=416)
          Spawn on: All datanodes
          -> Nested Loop (cost=0.00..45.99 rows=2 width=416)
            -> Streaming (type: REDISTRIBUTE) (cost=0.00..39.27 rows=2 width=258)
              Spawn on: All datanodes
              -> Nested Loop (cost=0.00..38.99 rows=2 width=258)
                -> Streaming (type: REDISTRIBUTE) (cost=0.00..32.28 rows=2 width=262)
                  Spawn on: All datanodes
                  -> Nested Loop (cost=0.00..32.00 rows=2 width=262)
                    -> Streaming (type: REDISTRIBUTE) (cost=0.00..25.28 rows=2
width=262)
                      Spawn on: All datanodes
                      -> Nested Loop (cost=0.00..25.00 rows=2 width=262)
                        -> Nested Loop (cost=0.00..21.64 rows=2 width=270)
                          -> Seq Scan on item (cost=0.00..13.36 rows=1
width=208)
                            Filter: ((i_current_price >= 35::numeric) AND
(i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND
(i_current_price <= 50::numeric) AND (i_color = ANY
('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                              -> Index Scan using store_sales_pkey on store_sales
                                (cost=0.00..8.27 rows=1 width=62)
                                  Index Cond: (ss_item_sk = item.i_item_sk)
                                  -> Index Only Scan using store_returns_pkey on
store_returns (cost=0.00..3.35 rows=1 width=8)
                                    Index Cond: ((sr_item_sk = store_sales.ss_item_sk) AND
(sr_ticket_number = store_sales.ss_ticket_number))
```

```

rows=1 width=8)
    -> Index Scan using customer_pkey on customer (cost=0.00..3.35
    Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
rows=1 width=4)
    -> Index Only Scan using promotion_pkey on promotion (cost=0.00..3.35
    Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
    -> Index Scan using store_pkey on store (cost=0.00..3.35 rows=1 width=166)
    Index Cond: (s_store_sk = store_sales.ss_store_sk)
rows=1 width=368)
    -> Index Scan using customer_address_pkey on customer_address ad2 (cost=0.00..3.35
    Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(34 rows)

```

6.10.2 Hint Specifying the Query Block Where the Hint Is Located

Description

This function allows users to use `@queryblock` in hints to implement block-level hint control. Users can specify the query block to which the hint takes effect. For example, you can specify the hint of an inner query block in the outer query block.

Syntax

Add **@queryblock** at the beginning of the hint parameter. **Hint_SEPC** is a specific hint.

```
Hint_SEPC([@queryblock])
```

Parameters

Hint_SEPC is the hint name, and **@queryblock** can be left empty. If **@queryblock** is left empty, the hint takes effect in the current query block declared by the hint. If **@queryblock** is left empty and **Hint_SPEC** has no parameter, use **Hint_SPEC** instead of **Hint_SPEC()**. Parentheses are unnecessary. The following describes how to name a query block and how to make a hint take effect. Some hints do not take effect only at the outermost layer and cannot be specified using **@queryblock**. For details, see the syntax description of each hint.

- Query the name of a query block.
 - Each query block must have a name, so as to accurately specify a hint. There are two naming methods: user-specified and system-specified.
 - You can use the `blockname` hint to specify the block to be queried. For details, see [Sublink Name Hints](#).
 - If no alias is specified for a query block, the default block name is automatically generated based on the processing sequence. Generally, the default alias of each query block consists of the first three letters of the query block name, `$`, and the number of the query block. For example, the alias of the first `SELECT` query block is `sel$1`. In pretty mode, you can use the `explain` method with a specified block name to view the name of the query block where the processing operator of each table is located. In distributed scenarios, only stream plans can be displayed in pretty mode. You can set **enable_fast_query_shipping** to generate stream plans.

```

gaussdb=# set explain_perf_mode = pretty;
SET
gaussdb=# set enable_fast_query_shipping = off;

```

```

SET
gaussdb=# explain (blockname on,costs off) select * from t1, (select c1 from t2 group by c1)
sub1 where t1.c1 = sub1.c1;
id |          operation          | Query Block
-----+-----+-----
 1 | -> Streaming (type: GATHER) | sel$1
 2 | -> Hash Join (3,4)         | sel$1
 3 | -> Seq Scan on t1@"sel$1"  | sel$1
 4 | -> Hash                    |
 5 | -> HashAggregate           | sel$2
 6 | -> Seq Scan on t2@"sel$2" | sel$2
(6 rows)

```

You can see that Seq Scan of **t2** is located in the **sel\$2** query block.

- **@queryblock** specifies the query block.

For the preceding example, if you want to modify the indexscan mode in **t2**, run the following command:

```

select /*+indexscan(@sel$2 t2) tablescan(t1)*/ * from t1, (select c1 from t2 group by c1) sub1 where
t1.c1 = sub1.c1;

```

Both indexscan and tablescan are scan hints. For details about scan hints, see [Scan Operation Hints](#). You can specify the hint of indexscan(**@sel\$2 t2**) in the **sel\$1** query block to move the hint to the **sel\$2** query block. The hint takes effect for **t2**. If the **sel\$2** query block is promoted **sel\$1** during subsequent rewriting, the hint is also promoted together to **sel\$1** and continues to take effect for **t2**.

```

gaussdb=# explain (blockname on,costs off) select /*+indexscan(@sel$2 t2) tablescan(t1)*/ * from t1,
(select c1 from t2 group by c1) sub1 where t1.c1 = sub1.c1;
id |          operation          | Query Block
-----+-----+-----
 1 | -> Streaming (type: GATHER) | sel$1
 2 | -> Hash Join (3,4)         | sel$1
 3 | -> Seq Scan on t1@"sel$1"  | sel$1
 4 | -> Hash                    |
 5 | -> HashAggregate           | sel$2
 6 | -> Index Only Scan using it2 on t2@"sel$2" | sel$2
(6 rows)

```

CAUTION

Sometimes, query rewriting in the optimizer phase expands some query blocks. As a result, the plan does not display related query blocks in the explain method. The hint specifies a query block based on the name of the query block before the optimizer phase. If a query block to be queried may be expanded in the planning phase, you can add the `no_expand` hint (see [Hints Specifying Not to Expand Subqueries](#)) to prevent it from being expanded.

1. The `sel$2` query block is a simple query. The optimizer performs query rewriting during subsequent processing, and `t1` is promoted to `sel$1` for processing. Therefore, the operation in the `sel$2` query block is not displayed in the plan.

```
gaussdb=# explain (blockname on, costs off) select * from t2, (select c1 from t1 where t1.c3 = 2) sub1
where t2.c1 = sub1.c1;
id | operation | Query Block
-----+-----+-----
1 | -> Streaming (type: GATHER) | sel$1
2 | -> Nested Loop (3,4) | sel$1
3 | -> Index Scan using it3 on t1@"sel$2" | sel$1
4 | -> Index Scan using it2 on t2@"sel$1" | sel$1
(4 rows)
```

2. The `sel$2` query block is a simple query. During subsequent processing, the optimizer skips query rewriting because of the `no_expand` hint, and `t1` is still processed in the original query block.

```
gaussdb=# explain (blockname on, costs off) select * from t2, (select /*+ no_expand*/ c1 from t1 where
t1.c3 = 2) sub1 where t2.c1 = sub1.c1;
id | operation | Query Block
-----+-----+-----
1 | -> Streaming (type: GATHER) | sel$1
2 | -> Nested Loop (3,4) | sel$1
3 | -> Index Scan using it3 on t1@"sel$2" | sel$2
4 | -> Index Scan using it2 on t2@"sel$1" | sel$1
(4 rows)
```

3. Because `t1` is processed in the `sel$2` query block after the `no_expand` hint is added, you can use `@sel$2` to specify the query block for the hint.

```
explain (blockname on, costs off) select /*+ tablescan(@sel$2 t1)*/ * from t2, (select c1 from t1 where
t1.c3 = 2) sub1 where t2.c1 = sub1.c1;
id | operation | Query Block
-----+-----+-----
1 | -> Streaming (type: GATHER) | sel$1
2 | -> Nested Loop (3,4) | sel$1
3 | -> Seq Scan on t1@"sel$2" | sel$1
4 | -> Index Scan using it2 on t2@"sel$1" | sel$1
(4 rows)
```

4. The query block number in the view depends on the sequence of the statement using the view. Therefore, do not use hints to specify query blocks when creating a view. The behavior is uncontrollable.

```
gaussdb=# create view v1 as select /*+ no_expand */ c1 from t1 where c1 in (select /*+ no_expand */ c1
from t2 where t2.c3=4 );
CREATE VIEW
gaussdb=# explain (blockname on, costs off) select * from v1;
id | operation | Query Block
-----+-----+-----
1 | -> Streaming (type: GATHER) | sel$1
2 | -> Seq Scan on t1@"sel$2" | sel$2
3 | -> Materialize [2, SubPlan 1] |
4 | -> Streaming(type: BROADCAST) |
5 | -> Seq Scan on t2@"sel$3" | sel$3
(5 rows)

Predicate Information (identified by plan id)
```

```
-----
2 --Seq Scan on t1@"sel$2"
  Filter: (hashed SubPlan 1)
5 --Seq Scan on t2@"sel$3"
  Filter: (c3 = 4)
(4 rows)
```

In this case, the statements in **v1** belong to **sel\$2** and **sel\$3**.

5. Some hints take effect only at the outermost layer and cannot be specified using **@queryblock**. For details, see the syntax description of each hint.

6.10.3 Hint Specifying the Query Block and Schema of a Table

Description

In a query, the table name can be duplicate in different query blocks and different schemas. Therefore, when specifying a table in a query, you can use a hint to specify the query block and schema where the table is located to avoid ambiguity. This function is applicable to all hints whose table names need to be specified.

Syntax

When specifying a table using a hint, specify a schema using a period (*schema*) and a query block using the at sign (*@queryblock*). Both the schema and query block can be left empty.

```
[schema.]relname[@queryblock]
```

Parameters

- **relname** indicates the name of the table in the query. If the table has an alias, use the alias first. In this case, **relname** is set to the alias. If the table name contains special characters, such as at sign (@) and period (.), **relname** must be enclosed in double quotation marks (") to avoid conflict with the declaration of the query block and schema names. For example, if the table name is **relnametest@1**, enter "**relnametest@1**".
- **schema** indicates the schema where the table is located. It can be left empty. If no schema is specified, the hint searches all schemas for **relname**.
- **queryblock** indicates the query block where the table is located. It can be left empty. If no query block is specified, the hint searches all query blocks for **relname**.

Example

1. **t1** of **sel\$2** is promoted to **sel\$1** for processing, and **t1** is unclear.

```
gaussdb=# explain(blockname on,costs off) select /*+ tablescan(t1)*/ * from t1, (select c2 from t1
where c1=1) tt1 where t1.c1 = tt1.c2;
WARNING: Error hint: TableScan(t1), relation name "t1" is ambiguous.
...
```
2. **t1@sel\$2** is specified to perform tablescan on **t1** of **sel\$2** (Filter: (c1 = 1)).

```
gaussdb=# explain(blockname on,costs off) select /*+ tablescan(t1@sel$2)*/ * from t1, (select c2 from
t1 where c1=1) tt1 where t1.c1 = tt1.c2;
id | operation | Query Block
-----+-----+-----
1 | -> Streaming (type: GATHER) | sel$1
2 | -> Nested Loop (3,5) | sel$1
```

```
3 | -> Streaming(type: REDISTRIBUTE) | sel$1
4 | -> Seq Scan on t1@"sel$2" | sel$1
5 | -> Index Scan using it1 on t1@"sel$1" | sel$1
(5 rows)

Predicate Information (identified by plan id)
-----
4 --Seq Scan on t1@"sel$2"
   Filter: (c1 = 1)
5 --Index Scan using it1 on t1@"sel$1"
   Index Cond: (c1 = public.t1.c2)
(4 rows)
```

6.10.4 Join Order Hints

Description

Specifies the join order and outer/inner tables.

Syntax

- Specify only the join order.

```
leading([@queryblock] join_table_list)
```

- Specify the join order and outer/inner tables. The outer/inner tables are specified by the outermost parentheses.

```
leading([@queryblock] (join_table_list))
```

Parameters

join_table_list specifies the tables to be joined. The values can be table names or table aliases. If a subquery is pulled up, the value can also be the subquery alias. Separate the values with spaces. You can add parentheses to specify the join priorities of tables.

For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.

NOTICE

A table name or alias can only be a string without a schema name.

An alias (if any) is used to represent a table.

To prevent semantic errors, tables in the list must meet the following requirements:

- The tables must exist in the query or its subquery to be pulled up.
- The table names must be unique in the query or subquery to be pulled up. If they are not, their aliases must be unique.
- A table appears only once in the list.
- An alias (if any) is used to represent a table.

For example:

leading(t1 t2 t3 t4 t5): t1, t2, t3, t4, and t5 are joined. The join sequence and outer/inner tables are not specified.

leading((t1 t2 t3 t4 t5)): t1, t2, t3, t4, and t5 are joined in sequence. The table on the right is used as the inner table in each join.

leading(t1 (t2 t3 t4) t5): First, t2, t3, and t4 are joined and the outer/inner tables are not specified. Then, the result is joined with t1 and t5, and the outer/inner tables are not specified.

leading((t1 (t2 t3 t4) t5)): First, t2, t3, and t4 are joined and the outer/inner tables are not specified. Then, the result is joined with t1, and (t2 t3 t4) is used as the inner table. Finally, the result is joined with t5, and t5 is used as the inner table.

leading((t1 (t2 t3) t4 t5) leading((t3 t2)): First, t2 and t3 are joined and t2 is used as the inner table. Then, the result is joined with t1, and (t2 t3) is used as the inner table. Finally, the result is joined with t4 and then t5, and the table on the right in each join is used as the inner table.

Example

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ leading((((store_sales store) promotion) item) customer) ad2) store_returns) leading((store
store_sales)*/ i_product_name product_name ...
```

First, **store_sales** and **store** are joined and **store_sales** is the inner table. Then, the result is joined with **promotion**, **item**, **customer**, **ad2**, and **store_returns** in sequence. The optimized plan is as follows:

```
WARNING: Duplicated or conflict hint: Leading(store_sales store), will be discarded.
```

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	2	880	78.79
2	-> HashAggregate	2	880	78.15
3	-> Nested Loop (4,24)	2	776	78.11
4	-> Streaming (type: REDISTRIBUTE)	2	784	71.39
5	-> Nested Loop (6,23)	2	784	70.75
6	-> Streaming (type: REDISTRIBUTE)	2	424	64.04
7	-> Nested Loop (8,22)	2	424	63.67
8	-> Streaming (type: REDISTRIBUTE)	2	424	56.96
9	-> Nested Loop (10,12)	2	424	56.59
10	-> Streaming (type: BROADCAST)	2	208	13.67
11	-> Seq Scan on item	1	208	13.36
12	-> Materialize	20	216	42.81
13	-> Hash Join (14,20)	20	216	42.79
14	-> Streaming (type: REDISTRIBUTE)	20	220	29.35
15	-> Hash Join (16,17)	20	220	27.73
16	-> Seq Scan on store	20	166	13.13
17	-> Hash	21	62	14.31
18	-> Streaming (type: REDISTRIBUTE)	20	62	14.31
19	-> Seq Scan on store_sales	20	62	13.13
20	-> Hash	21	4	13.13
21	-> Seq Scan on promotion	20	4	13.13
22	-> Index Scan using customer_pkey on customer	1	8	3.35
23	-> Index Scan using customer_address_pkey on customer_address ad2	1	368	3.35
24	-> Index Only Scan using store_returns_pkey on store_returns	1	8	3.35

(24 rows)

For details about the warning at the top of the plan, see [Hint Errors, Conflicts, and Other Warnings](#).

6.10.5 Join Operation Hints

Function

These hints specify the join method, which can be nested loop join, hash join, or merge join.

Syntax

```
[no] nestloop|hashjoin|mergejoin([@queryblock] table_list)
```

Parameter Description

- For details about `@queryblock`, see [Hint Specifying the Query Block Where the Hint Is Located](#). `@queryblock` can be omitted, indicating that the hint takes effect in the current query block.
- `no` indicates that the specified hint will not be used for a join.
- `table_list` specifies the tables to be joined. The values are the same as those of [join_table_list](#) but contain no parentheses.

For example:

no nestloop(t1 t2 t3): `nestloop` is not used for joining `t1`, `t2`, and `t3`. The three tables may be joined in either of the two ways: Join `t2` and `t3`, and then `t1`; join `t1` and `t2`, and then `t3`. This hint takes effect only for the last join. If necessary, you can hint other joins. For example, you can add `no nestloop(t2 t3)` to join `t2` and `t3` first and to forbid the use of `nestloop`.

Example

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

`nestloop` is used for the last join between `store_sales`, `store_returns`, and `item`. The optimized plan is as follows:

id	operation	E-rows	E-width	E-costs
1	-> HashAggregate	1	880	53.76
2	-> Streaming (type: GATHER)	2	880	53.76
3	-> HashAggregate	2	880	53.11
4	-> Nested Loop (5,20)	2	776	53.07
5	-> Streaming(type: REDISTRIBUTE)	2	416	46.36
6	-> Nested Loop (7,19)	2	416	45.99
7	-> Streaming(type: REDISTRIBUTE)	2	258	39.27
8	-> Nested Loop (9,18)	2	258	38.99
9	-> Streaming(type: REDISTRIBUTE)	2	262	32.28
10	-> Nested Loop (11,17)	2	262	32.00
11	-> Streaming(type: REDISTRIBUTE)	2	262	25.28
12	-> Nested Loop (13,16)	2	262	25.00
13	-> Nested Loop (14,15)	2	270	21.64
14	-> Seq Scan on item	1	208	13.36
15	-> Index Scan using store_sales_pkey on store_sales	1	62	8.27
16	-> Index Only Scan using store_returns_pkey on store_returns	1	8	3.35
17	-> Index Scan using customer_pkey on customer	1	8	3.35
18	-> Index Only Scan using promotion_pkey on promotion	1	4	3.35
19	-> Index Scan using store_pkey on store	1	166	3.35
20	-> Index Scan using customer_address_pkey on customer_address ad2	1	368	3.35

(20 rows)

6.10.6 Rows Hints

Description

These hints specify the number of rows in an intermediate result set. Both absolute values and relative values are supported.

Syntax

```
rows( [@queryblock] table_list #|+|-* const)
```

Parameters

- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.
- **#**, **+**, **-**, and ***** are operators used for hinting the estimation. **#** indicates that the original estimation is used without any calculation. **+**, **-**, and ***** indicate that the original estimation is calculated using these operators. The minimum calculation result is 1. **table_list** specifies the tables to be joined. The values are the same as those of **table_list** in [Join Operation Hints](#).
- **const** can be any non-negative number and supports scientific notation.

For example:

rows(t1 #5): The result set of **t1** is five rows.

rows(t1 t2 t3 *1000): The result set of joined **t1**, **t2**, and **t3** is multiplied by 1000.

Suggestion

- The hint using ***** for two tables is recommended. This hint will be triggered if the two tables appear on two sides of a join. For example, if the hint is **rows(t1 t2 * 3)**, the join result of **(t1 t3 t4)** and **(t2 t5 t6)** will be multiplied by 3 because **t1** and **t2** appear on both sides of the join.
- **rows** hints can be specified for the result sets of a single table, multiple tables, function tables, and subquery scan tables.

Example

Hint the query plan in [Examples](#) as follows:

```
explain  
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```

Multiply the result set of joined **store_sales** and **store_returns** by 50. The optimized plan is as follows.

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	4	880	58.44
2	-> HashAggregate	4	880	57.15
3	-> Nested Loop (4,20)	5	776	57.07
4	-> Streaming(type: REDISTRIBUTE)	2	784	50.34
5	-> Nested Loop (6,19)	2	784	49.71
6	-> Streaming(type: REDISTRIBUTE)	2	424	43.00
7	-> Nested Loop (8,18)	2	424	42.63
8	-> Streaming(type: REDISTRIBUTE)	2	266	35.91
9	-> Nested Loop (10,17)	2	266	35.63
10	-> Streaming(type: REDISTRIBUTE)	2	270	28.92
11	-> Nested Loop (12,16)	2	270	28.63
12	-> Streaming(type: REDISTRIBUTE)	2	270	21.92
13	-> Nested Loop (14,15)	2	270	21.64
14	-> Seq Scan on item	1	208	13.36
15	-> Index Scan using store_sales_pkey on store_sales	1	62	8.27
16	-> Index Scan using customer_pkey on customer	1	8	3.35
17	-> Index Only Scan using promotion_pkey on promotion	1	4	3.35
18	-> Index Scan using store_pkey on store	1	166	3.35
19	-> Index Scan using customer_address_pkey on customer_address ad2	1	368	3.35
20	-> Index Only Scan using store_returns_pkey on store_returns	1	8	3.35

(20 rows)

The estimation value after the hint in row 11 is **360**, and the original value is rounded off to 7.

6.10.7 Stream Operation Hints

Function

These hints specify a streaming operation, which can be **broadcast** or **redistribute**. You can also directly specify a method to generate a gather plan.

Syntax

```
[no] broadcast|redistribute( [@queryblock] table_list)
gather( [@queryblock] REL|JOIN|ALL)
```

Parameter Description

- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.
- **broadcast** and **redistribute**
 - **no** specifies that the specified hint will not be used for a stream operation.
 - *table_list* specifies the table on which a stream operation is to be performed or the tables to be joined. For details, see [Parameter Description](#).
- **gather**

The gather hint can specify the following plan generation modes:

 - **REL**: Only the gather path based on the base table is generated, and then the remaining plan is executed on the CN.
 - **JOIN**: A join-based gather path is generated as much as possible and is added to the join subplan that can be pushed down (the join subplan does not contain the redistribution node), and the remaining plan is executed on the CN. For a join plan that requires node redistribution, such a join-based gather path cannot be generated. Instead, a base table-based gather path is generated.

CAUTION

After **Hint(JOIN)** is specified, the plan expected by **Hint(JOIN)** cannot be generated if the distribution table and replication table are joined, because the optimizer has found a better plan for replacement.

- **ALL:** The **Gather Rel** or **Gather Join** path is selected based on the optimal mode.

Example

Hint the query plan in **Examples** as follows:

```
explain
select /*+ no redistribute(store_sales store_returns item store) leading(((store_sales store_returns item
store) customer)) */ i_product_name product_name ...
```

In the original plan, the join result of **store_sales**, **store_returns**, **item**, and **store** is redistributed before it is joined with **customer**. After the hinting, the redistribution is disabled and the join order is retained. The optimized plan is as follows:

id	operation	E-rows	E-width	E-costs
1	-> HashAggregate	1	880	62.28
2	-> Streaming (type: GATHER)	2	880	62.28
3	-> HashAggregate	2	880	61.63
4	-> Nested Loop (5,21)	2	776	61.59
5	-> Streaming (type: REDISTRIBUTE)	2	416	54.88
6	-> Nested Loop (7,20)	2	416	54.51
7	-> Streaming (type: REDISTRIBUTE)	2	420	47.79
8	-> Nested Loop (9,17)	2	420	47.42
9	-> Nested Loop (10,16)	2	420	32.00
10	-> Streaming (type: REDISTRIBUTE)	2	262	25.28
11	-> Nested Loop (12,15)	2	262	25.00
12	-> Nested Loop (13,14)	2	270	21.64
13	-> Seq Scan on item	1	208	13.36
14	-> Index Scan using store_sales_pkey on store_sales	1	62	8.27
15	-> Index Only Scan using store_returns_pkey on store_returns	1	8	3.35
16	-> Index Scan using store_pkey on store	1	166	3.35
17	-> Materialize	40	8	15.23
18	-> Streaming (type: BROADCAST)	40	8	15.18
19	-> Seq Scan on customer	20	8	13.13
20	-> Index Only Scan using promotion_pkey on promotion	1	4	3.35
21	-> Index Scan using customer_address_pkey on customer_address ad2	1	368	3.35

(21 rows)

Specify the gather hint for a statement.

1. Generate the gather plan **/* + GATHER(REL)*/** based on the base table.

```
gaussdb=# explain select /*+ GATHER(REL)*/ from t1, t2, t3 where t1.c2 = t2.c2 and t2.c2 = t3.c2;
```

id	operation	E-rows	E-width	E-costs
1	-> Hash Join (2,8)	20	36	44.10
2	-> Hash Join (3,5)	20	24	29.22
3	-> Streaming (type: GATHER)	20	12	14.35
4	-> Seq Scan on t1	20	12	13.13
5	-> Hash	20	12	14.35
6	-> Streaming (type: GATHER)	20	12	14.35
7	-> Seq Scan on t2	20	12	13.13
8	-> Hash	20	12	14.35
9	-> Streaming (type: GATHER)	20	12	14.35
10	-> Seq Scan on t3	20	12	13.13

(10 rows)

Predicate Information (identified by plan id)

1	--Hash Join (2,8)	Hash Cond: (t1.c2 = t3.c2)
2	--Hash Join (3,5)	Hash Cond: (t1.c2 = t2.c2)

(4 rows)

2. Generate the join gather plan **/*+ GATHER(REL)*** that can be pushed down.

```
gaussdb=# explain select /*+ GATHER(JOIN)*/* from t1, t2, t3 where t1.c1 = t2.c1 and t2.c2 = t3.c2;
```

id	operation	E-rows	E-width	E-costs
1	-> Hash Join (2,7)	20	36	42.37
2	-> Streaming (type: GATHER)	20	24	27.49
3	-> Hash Join (4,5)	20	24	26.56
4	-> Seq Scan on t1	20	12	13.13
5	-> Hash	21	12	13.13
6	-> Seq Scan on t2	20	12	13.13
7	-> Hash	20	12	14.35
8	-> Streaming (type: GATHER)	20	12	14.35
9	-> Seq Scan on t3	20	12	13.13

(9 rows)

Predicate Information (identified by plan id)

```
1 --Hash Join (2,7)
  Hash Cond: (t2.c2 = t3.c2)
3 --Hash Join (4,5)
  Hash Cond: (t1.c1 = t2.c1)
```

(4 rows)

3. Generate the gather plan **/*+ GATHER(ALL)*** based on the optimal mode.

The **GATHER(REL)** or **GATHER(JOIN)** path is selected based on the optimal mode and rules.

```
gaussdb=# explain select /*+ GATHER(ALL)*/* from t1, t2, t3 where t1.c1 = t2.c1 and t2.c2 = t3.c2;
```

id	operation	E-rows	E-width	E-costs
1	-> Hash Join (2,7)	20	36	42.37
2	-> Streaming (type: GATHER)	20	24	27.49
3	-> Hash Join (4,5)	20	24	26.56
4	-> Seq Scan on t1	20	12	13.13
5	-> Hash	21	12	13.13
6	-> Seq Scan on t2	20	12	13.13
7	-> Hash	20	12	14.35
8	-> Streaming (type: GATHER)	20	12	14.35
9	-> Seq Scan on t3	20	12	13.13

(9 rows)

Predicate Information (identified by plan id)

```
1 --Hash Join (2,7)
  Hash Cond: (t2.c2 = t3.c2)
3 --Hash Join (4,5)
  Hash Cond: (t1.c1 = t2.c1)
```

(4 rows)

6.10.8 Scan Hints

Description

These hints specify a scan operation, which can be **tablescan**, **indexscan**, **indexonlyscan** or **gsi**.

Syntax

```
[no] tablescan|indexscan|indexonlyscan|gsi( [@queryblock] table [index])
```

Parameters

- **no** indicates that the specified hint will not be used for a join.

- For details about `@queryblock`, see [Hint Specifying the Query Block Where the Hint Is Located](#). `@queryblock` can be omitted, indicating that the hint takes effect in the current query block.
- `table` specifies the table to be scanned. You can specify only one table. Use a table alias (if any) instead of a table name.
- `index` indicates the index for `indexscan`, `indexonlyscan` or `gsi`. You can specify only one index.

 NOTE

- Index scan, index-only scan, and GSI hints can be used only when the hint index belongs to the hint table.
- Scan operation hints can be used for row-store tables and subquery tables.
- The index-only scan plan can be generated by the index scan hint, but the index-only hint can generate only the index-only plan.
- When index scan is compatible with index-only scan, some plan changes may occur. For example, `cost_model_version` is added for escape. This parameter can be used to determine whether index scan is compatible with index-only scan. Index scan is compatible with index-only scan when the parameter value is greater than 2 or equal to 0.
- If a GSI can be used for a query, using the GSI hint will generate an index-only scan query plan that uses the GSI, but not a query plan that uses a common index.
- If no queryblock, table, or index is specified for a GSI hint, the GSI hint takes effect across queryblocks.
- Valid GSIs can be directly queried on DNs.
- In the current version, GSIs cannot be used for table access by index row ID.

Example

To specify an index-based hint for a scan, create an index named `i` on the `i_item_sk` column of the `item` table.

```
create index i on item(i_item_sk);
```

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```

`item` is scanned based on an index. The optimized plan is as follows.

```
WARNING: Error hint: IndexScan(item i), index "i" doesn't exist.
```

id	operation	E-rows	E-width	E-costs
1	-> HashAggregate	1	880	53.76
2	-> Streaming (type: GATHER)	2	880	53.76
3	-> HashAggregate	2	880	53.11
4	-> Nested Loop (5,20)	2	776	53.07
5	-> Streaming (type: REDISTRIBUTE)	2	416	46.36
6	-> Nested Loop (7,19)	2	416	45.99
7	-> Streaming (type: REDISTRIBUTE)	2	258	39.27
8	-> Nested Loop (9,18)	2	258	38.99
9	-> Streaming (type: REDISTRIBUTE)	2	262	32.28
10	-> Nested Loop (11,17)	2	262	32.00
11	-> Streaming (type: REDISTRIBUTE)	2	262	25.28
12	-> Nested Loop (13,16)	2	262	25.00
13	-> Nested Loop (14,15)	2	270	21.64
14	-> Seq Scan on item	1	208	13.36
15	-> Index Scan using store_sales_pkey on store_sales	1	62	8.27
16	-> Index Only Scan using store_returns_pkey on store_returns	1	8	3.35
17	-> Index Scan using customer_pkey on customer	1	8	3.35
18	-> Index Only Scan using promotion_pkey on promotion	1	4	3.35
19	-> Index Scan using store_pkey on store	1	166	3.35
20	-> Index Scan using customer_address_pkey on customer_address ad2	1	368	3.35

(20 rows)

6.10.9 Sublink Name Hints

Description

These hints specify the name of a sublink block.

Syntax

```
blockname ( [@queryblock] table)
```

Parameters

- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.
- **table** specifies the name you have specified for a sublink block.

NOTE

- The **blockname** hint is used by an outer query only when the corresponding sublink is not pulled up. Currently, only the **Agg** equivalent join, **IN**, and **EXISTS** sublinks can be pulled up. This hint is usually used together with the hints described in the previous sections.
- The subquery after the **FROM** keyword is hinted by using the subquery alias. In this case, **blockname** becomes invalid.
- If a sublink contains multiple tables, the tables will be joined with the outer-query tables in a random sequence after the sublink is pulled up. In this case, **blockname** also becomes invalid.

Examples

```
explain select /*+nestloop(store_sales tt) */ * from store_sales where ss_item_sk in (select /*+blockname(tt)*/ i_item_sk from item group by 1);
```

tt indicates the sublink block name. After being pulled up, the sublink is joined with the outer-query table **store_sales** by using **nestloop**. The optimized plan is as follows.

```
gaussdb=# explain select /*+nestloop(store_sales tt) */ * from store_sales wh
id |          operation          | E-rows | E-width | E-costs
---+-----+-----+-----+-----
 1 | -> Streaming (type: GATHER) |      20 |      212 | 30.02
 2 | -> Nested Loop Semi Join (3, 4) |      20 |      212 | 28.47
 3 | -> Seq Scan on store_sales |      20 |      212 | 13.13
 4 | -> Materialize |      20 |         4 | 13.41
 5 | -> HashAggregate |      20 |         4 | 13.26
 6 | -> Seq Scan on item |      20 |         4 | 13.13
(6 rows)
```

CAUTION

When the blockname hint is specified using **@queryblock** instead of taking effect in the current query block, for example, **blockname(@sel\$2 new_qb_name)**, other hints cannot be specified using **@new_qb_name**. In this case, **new_qb_name** is only used as the name of the sublink and can be specified using hints.

- The following specifies block **bn2** using **blockname(@sel\$2 bn2)**. As a result, **TableScan(@bn2 t2)** cannot find the queryblock by using **@bn2**. The query block should be specified by using **@sel\$2**. The following specifies block **bn3** using the **blockname(bn3)** hint. The hint takes effect in the current query block and changes the name of the query block. Therefore, **tablescan(@bn3 t3@bn3)** can find the query block specified by using **@bn3**.

```
gaussdb=# explain select /*+ blockname(@sel$2 bn2) tablescan(@bn2 t2) tablescan(@sel$2 t2@bn2)
indexscan(@sel$2 t2@sel$2) tablescan(@bn3 t3@bn3)*/ c2 from t1 where c1 in ( select /*+ */t2.c1
from t2 where t2.c2 = 1 group by 1) and c3 in ( select /*+ blockname(bn3)*/t3.c3 from t3 where t3.c2
= 1 group by 1);
```

```
WARNING: hint: TableScan(@bn2 t2) does not match any query block
```

```
WARNING: Error hint: TableScan(@"sel$2" t2@bn2), relation name "t2@bn2" is not found.
```

- The following specifies the sublink block **bn2** by using **blockname(@sel\$2 bn2)**. When the sublink is promoted, **hashjoin(t1 bn2)** can be used to specify the operation of the promoted sublink.

```
gaussdb=# explain select /*+ blockname(@sel$2 bn2) hashjoin(t1 bn2) nestloop(t1 bn3) nestloop(t1
sel$3)*/ c2 from t1 where c1 in ( select /*+ */t2.c1 from t2 where t2.c2 = 1 group by 1) and c3 in
( select /*+ blockname(bn3)*/t3.c3 from t3 where t3.c2 = 1 group by 1);
```

```
WARNING: Duplicated or conflict hint: NestLoop(t1 "sel$3"), will be discarded.
```

6.10.10 Skew Hints

Description

Specifies redistribution keys containing skew data and skew values, and are used to optimize redistribution involving Join or HashAgg.

Syntax

- Specify single-table skew.
`skew([@queryblock] table (column) [(value)])`
- Specify intermediate result skew.
`skew([@queryblock] (join_rel) (column) [(value)])`

Parameters

- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.
- table** specifies the table where skew occurs.
- join_rel** specifies two or more joined tables. For example, **(t1 t2)** indicates that the result of joining **t1** and **t2** tables contains skew data.
- column** specifies one or more columns where skew occurs.
- value** specifies one or more skew values.

 NOTE

- Skew hints are used only if redistribution is required and the specified skew information matches the redistribution information.
- Skew hints are controlled by the GUC parameter **skew_option**. If the parameter is disabled, skew hints cannot be used for solving skew.
- Currently, skew hints support only the table relationships of the ordinary table and subquery types. Hints can be specified for base tables, subqueries, and **WITH ... AS** clauses. Unlike other hints, a subquery can be used in skew hints regardless of whether it is pulled up.
- Use an alias (if any) to specify a table where data skew occurs.
- You can use a name or an alias to specify a skew column as long as it is not ambiguous. The columns in skew hints cannot be expressions. If data skew occurs in the redistribution that uses an expression as a redistribution key, set the redistribution key as a new column and specify the column in skew hints.
- The number of skew values must be an integer multiple of the number of columns. Skew values must be grouped based on the column sequence, with each group containing a maximum of 10 values. You can specify duplicate values to group skew columns having different number of skew values. For example, the **c1** and **c2** columns of the **t1** table contain skew data. The skew value of the **c1** column is **a1**, and the skew values of the **c2** column are **b1** and **b2**. In this case, the skew hint is **skew(t1 (c1 c2) ((a1 b1)(a1 b2)))**. **(a1 b1)** is a value group, where **NULL** is allowed as a skew value. Each hint can contain a maximum of 10 groups and the number of groups should be an integer multiple of the number of columns.
- In the redistribution optimization of Join, a skew value must be specified for skew hints. The skew value can be left empty for HashAgg.
- If multiple tables, columns, or values are specified, separate items of the same type with spaces.
- The type of skew values cannot be forcibly converted in hints. To specify a string, enclose it with single quotation marks (' ').

Example:

- Specify single-table skew.

Each skew hint describes the skew information of one table relationship. To describe the skews of multiple table relationships in a query, specify multiple skew hints.

Skew hints have the following formats:

- One skew value in one column: **skew(t (c1) (v1))**
Description: The **v1** value in the **c1** column of the **t** table relationship causes skew in query execution.
- Multiple skew values in one column: **skew(t (c1) (v1 v2 v3 ...))**
Description: Values including **v1**, **v2**, and **v3** in the **c1** column of the **t** table relationship cause skew in query execution.
- Multiple columns, each having one skew value: **skew(t (c1 c2) (v1 v2))**
Description: The **v1** value in the **c1** column and the **v2** value in the **c2** column of the **t** table relationship cause skew in query execution.
- Multiple columns, each having multiple skew values: **skew(t (c1 c2) ((v1 v2) (v3 v4) (v5 v6) ...))**
Description: Values including **v1**, **v3**, and **v5** in the **c1** column and values including **v2**, **v4**, and **v6** in the **c2** column of the **t** table relationship cause skew in query execution.

NOTICE

In the last format, parentheses for skew value groups can be omitted, for example, **skew(t (c1 c2) (v1 v2 v3 v4 v5 v6 ...))**. In a skew hint, either use parentheses for all skew value groups or for none of them.

Otherwise, a syntax error will be generated. For example, **skew(t (c1 c2) (v1 v2 v3 v4 (v5 v6) ...))** will generate an error.

- Specify intermediate result skew.

If data skew does not occur in base tables but in an intermediate result during query execution, specify skew hints of the intermediate result to solve the skew. `skew((t1 t2) (c1) (v1))`

Description: Data skew occurs after the table relationships **t1** and **t2** are joined. The **c1** column of the **t1** table contains skew data and its skew value is **v1**.

c1 can exist only in a table relationship of **join_rel**. If there is another column having the same name, use aliases to avoid ambiguity.

Suggestion

- For a multi-level query, write the hint on the layer where data skew occurs.
- For a listed subquery, you can specify the subquery name in a hint. If you know data skew occurs on which base table, directly specify the table.
- Aliases are preferred when you specify a table or column in a hint.

6.10.11 Parameterized Path Hint

Function

Specifies the parameterized path and the conditional predicate pushdown method.

Syntax

```
predpush( [@queryblock] src1 src2)  
predpush( [@queryblock] src, dest)
```

Parameter Description

- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.
- **src**, **src1**, and **src2** indicate the set of candidates tables pushed down by **predpush**.
- **dest** indicates the specified destination table pushed down by **predpush**.
- If **predpush** does not contain commas (,), all tables are candidates table. If **predpush** contains commas (,), both candidates tables and destination tables are specified.

 NOTE

Use the **predpush** hint to move the filter expression as close to the data source as possible to optimize the query.

- Before using the **predpush** hint, ensure that the **rewrite_rule** GUC parameter contains the **PREDPUSH|REDPUSHFORCE|PREDPUSHNORMAL** option.
- **subquery_block** can also be a view or materialized view.

Examples

Use the **predpush** hint to improve the statement execution efficiency. Example:

```
create table pt2(a int, b int);
create table pt3(a int, b int);
create table pt4(a int, b int);
create index t4_a_idx on pt4(a);
create index t3_a_idx on pt3(a, b);
create index t2_a_idx on pt2(a);
set rewrite_rule='predpushforce';
set enable_fast_query_shipping = off;
set explain_perf_mode=pretty;
gaussdb=# explain (costs off) SELECT /*+PREDPUSH(pt2 st3) */ *
FROM pt2,
      (SELECT /*+ indexscan(pt3) indexscan(pt4) */sum(pt3.b), pt3.a FROM pt3, pt4 where pt3.a = pt4.a
GROUP BY pt3.a) st3
WHERE st3.a = pt2.a;
id | operation
-----+-----
 1 | -> Streaming (type: GATHER)
 2 | -> Nested Loop (3,4)
 3 | -> Seq Scan on pt2
 4 | -> HashAggregate
 5 | -> Nested Loop (6,7)
 6 | -> Index Only Scan using t3_a_idx on pt3
 7 | -> Index Only Scan using t4_a_idx on pt4
(7 rows)
Predicate Information (identified by plan id)
-----
 6 --Index Only Scan using t3_a_idx on pt3
   Index Cond: (a = pt2.a)
 7 --Index Only Scan using t4_a_idx on pt4
   Index Cond: (a = pt2.a)
(4 rows)
```

If the **predpush** hint is not used, pt3 and pt4 in the subquery are not processed outside the query block before being joined. As a result, the returned result set is large, causing performance waste.

However, as shown in the preceding plan, after the **predpush** hint is used, condition filtering is performed on pt3 and pt4 based on pt2 before they are joined. The result set returned after joining is small, which effectively improves the performance.

6.10.12 Hint Errors, Conflicts, and Other Warnings

Plan hints change an execution plan. You can run **EXPLAIN** to view the changes.

Hints containing errors are invalid and do not affect statement execution. The errors will be displayed in different ways based on statement types. Hint errors in an **EXPLAIN** statement are displayed as a warning on the interface. Hint errors in other statements will be recorded in debug1-level logs containing the **PLANHINT** keyword.

Hint error types are as follows:

- Syntax errors

An error will be reported if the syntax tree fails to be reduced. The No. of the row generating an error is displayed in the error details.

For example, the hint keyword is incorrect, no table or only one table is specified in the **leading** or **join** hint, or no tables are specified in other hints. The parsing of a hint is terminated immediately after a syntax error is detected. Only the hints that have been parsed successfully are valid.

Example:

```
leading((t1 t2)) nestloop(t1) rows(t1 t2 #10)
```

The syntax of **nestloop(t1)** is wrong and its parsing is terminated. Only **leading(t1 t2)** that has been successfully parsed before **nestloop(t1)** is valid.

- Semantic errors

- An error will be reported if the specified tables do not exist, multiple tables are found based on the hint setting, or a table is used more than once in the **leading** or **join** hint.
- An error will be reported if the index specified in a scan hint does not exist.
- If multiple tables with the same name exist after a subquery is pulled up and some of them need to be hinted, add aliases for them to avoid name duplication.

- Duplicated or conflicted hints

If hint duplication or conflicts occur, only the first hint takes effect. A message will be displayed to describe the situation.

- Hint duplication indicates that a hint is used more than once in the same query, for example, **nestloop(t1 t2) nestloop(t1 t2)**.
- A hint conflict indicates that the functions of two hints with the same table list conflict with each other.

For example, if **nestloop (t1 t2) hashjoin (t1 t2)** is used, **hashjoin (t1 t2)** becomes invalid. **nestloop(t1 t2)** does not conflict with **no mergejoin(t1 t2)**.

NOTICE

The table list in the **leading** hint is disassembled. For example, **leading ((t1 t2 t3))** will be disassembled as **leading((t1 t2)) leading(((t1 t2) t3))**, which will conflict with **leading((t2 t1))** (if any). In this case, the latter **leading(t2 t1)** becomes invalid. If two hints use duplicated table lists and only one of them has the specified outer/inner table, the one without a specified outer/inner table becomes invalid.

- A hint becomes invalid after a sublink is pulled up.

In this case, a message will be displayed. Generally, such invalidation occurs when a sublink contains multiple tables to be joined. After the sublink is pulled up, the tables will not be join members.

- Unsupported column types

- Skew hints are specified to optimize redistribution. They will be invalid if their corresponding columns do not support redistribution.
- Hints are not used.
 - If a **hashjoin** or **mergejoin** hint is specified for non-equivalent joins, it will not be used.
 - If an **indexscan** or **indexonlyscan** hint is specified for a table that does not have an index, it will not be used.
 - The GSI hint is used for tables that do not contain GSIs.
 - If an IndexScan, IndexOnlyScan or GSI hint is specified for a full-table scan, it will not be used. Generally, index paths are generated only when filter conditions are used on index columns. Indexes are not used during a full table scan.
 - The specified IndexOnlyScan hint is used only when the output column contains only indexes.
 - The GSI hint is used only when queries can be pushed down to GSIs.
 - In equivalent joins, only the joins containing equivalence conditions are valid. Therefore, the **leading**, **join**, and **rows** hints specified for the joins without an equivalence condition will not be used. For example, **t1**, **t2**, and **t3** are to be joined, and the join between **t1** and **t3** does not contain an equivalence condition. In this case, **leading(t1 t3)** will not be used.
 - To generate a streaming plan, if the distribution key of a table is the same as its join key, **redistribute** specified for this table will not be used. If the distribution key and join key are different for this table but the same for the other table in the join, **redistribute** specified for this table will be used but **broadcast** will not.
 - If no sublink is pulled up, the specified **blockname** hint will not be used.
 - Skew hints are not used possibly because:
 - The plan does not require redistribution.
 - The columns specified by hints contain distribution keys.
 - Skew information specified in hints is incorrect or incomplete, for example, no value is specified for join optimization.
 - Skew optimization is disabled by GUC parameters.

6.10.13 GUC Parameter Hints

Description

Sets GUC parameters related to query optimization. The settings take effect during the query execution. For details about the application scenarios of hints, see the description of each GUC parameter.

Syntax

```
set( [@queryblock] param value)
```

Parameters

- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block. This hint takes effect only when it specifies the outermost queryblock.
- **param** indicates the parameter name.
- **value** indicates the value of a parameter.
- Currently, the following parameters can be set and take effect by using hints:
 - Boolean type
enable_bitmapscan, enable_hashagg, enable_hashjoin, enable_indexscan, enable_indexonlyscan, enable_gscan, enable_material, enable_mergejoin, enable_nestloop, enable_index_nestloop, enable_seqscan, enable_sort, enable_tidscan, enable_stream_operator, enable_stream_recursive, enable_broadcast, enable_fast_query_shipping, enable_trigger_shipping, enable_remotejoin, enable_remotegroup, enable_remotelimit, enable_remotesort, and enable_inner_unique_opt
 - Integer type
best_agg_plan and query_dop
 - Floating point type
cost_weight_index, default_limit_rows, seq_page_cost, random_page_cost, cpu_tuple_cost, cpu_index_tuple_cost, cpu_operator_cost, and effective_cache_size
 - Character string type
node_name
By setting **node_name**, you can deliver the current SQL statement to the DN corresponding to **node_name** for execution.

Example:

```
select /*+ set(node_name datanode1) */ from table_name;
```

In the preceding command, **datanode1** indicates the name of the DN queried from the **pgxc_node** system catalog (without quotation marks), and **table_name** indicates the table name. This query is directly performed on **datanode1**.

NOTICE

- **node_name** can be set only by using the SELECT statement. If it is set by using other statements, it does not take effect.
- **node_name** can only be set to the name of a DN and cannot be set to the name of a CN.
- **node_name** cannot be modified by using the SET statement and can only be used in plan hints.
- **node_name** cannot be modified by using **gs_guc**.
- **node_name** supports only simple query statements and does not support complex query statements (such as UNION and UNION ALL), subqueries, and multi-table associations.
- This operation can be performed by common users.
- This operation cannot be performed together with row-level security. If they are performed together, an error will be reported.

NOTE

- If you set a parameter that is not in the whitelist and the parameter value is invalid or the hint syntax is incorrect, the query execution is not affected. Run **explain(verbose on)**. An error message is displayed, indicating that hint parsing fails.
- The GUC parameter hint takes effect only in the outermost query. That is, the GUC parameter hint in the subquery does not take effect.
- The GUC parameter hint in the view definition does not take effect.
- In the CREATE TABLE ... AS ... statement, the outermost GUC parameter hint takes effect.

6.10.14 Hints for Selecting the Custom Plan or Generic Plan

Function

For query statements and DML statements executed in PBE mode, the optimizer generates a custom plan or generic plan based on factors such as rules, costs, and parameters. You can use the hint of **use_cplan** or **use_gplan** to specify the plan to execute.

Syntax

- To select the custom plan, run the following statement:
`use_cplan`
- To select the generic plan, run the following statement:
`use_gplan`

NOTE

- For SQL statements that are executed in non-PBE mode, setting this hint does not affect the execution mode.
- This hint has a higher priority than cost-based selection and the **plan_cache_mode** parameter. That is, this hint does not take effect for statements for which **plan_cache_mode** cannot be forcibly set to specify an execution mode.

Example

Forcibly use the custom plan.

```
set enable_fast_query_shipping = off;
create table t (a int, b int, c int);
prepare p as select /*+ use_cplan */ * from t where a = $1;
explain execute p(1);
```

In the following plan, the filtering condition is the actual value of the input parameter, that is, the plan is a custom plan.

```
QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.06..13.26 rows=1 width=12)
  Node/s: datanode1
  -> Seq Scan on t (cost=0.00..13.16 rows=1 width=12)
      Filter: (a = 1)
(4 rows)
```

Forcibly use the generic plan.

```
deallocate p;
prepare p as select /*+ use_gplan */ * from t where a = $1;
explain execute p(1);
```

In the following plan, the filtering condition is the input parameter to be added, that is, the plan is a generic plan.

```
QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.06..13.26 rows=1 width=12)
  Node/s: All datanodes
  -> Seq Scan on t (cost=0.00..13.16 rows=1 width=12)
      Filter: (a = $1)
(4 rows)
```

6.10.15 Hints Specifying Not to Expand Subqueries

Function

When the database optimizes the query logic, some subqueries can be promoted to the upper layer to avoid nested execution. However, for some subqueries that have a low selection rate and can use indexes to filter access pages, nested execution does not cause too much performance deterioration, while after the promotion, the query search scope is expanded, which may cause performance deterioration. In this case, you can use the **no_expand** hint for debugging. This hint is not recommended in most cases.

Syntax

```
no_expand[(@queryblock)]
```

Parameter Description

For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block. If it is not specified, **no_expand** does not have parentheses ().

Example

Normal query execution:

```
explain select * from t1 where t1.c1 in (select t2.c1 from t2);
```

Plan

```

-----
QUERY PLAN
-----
Streaming (type: GATHER) (cost=16.98..34.28 rows=40 width=12)
  Node/s: All datanodes
  -> Hash Semi Join (cost=16.36..32.72 rows=40 width=12)
      Hash Cond: (t1.c1 = t2.c1)
      -> Seq Scan on t1 (cost=0.00..16.16 rows=40 width=12)
      -> Hash (cost=16.16..16.16 rows=40 width=4)
          -> Seq Scan on t2 (cost=0.00..16.16 rows=40 width=4)
(7 rows)

```

After `no_expand` is added:

```
explain select * from t1 where t1.c1 in (select /*+ no_expand*/ t2.c1 from t2);
```

Plan

```

-----
QUERY PLAN
-----
Streaming (type: GATHER) (cost=17.38..34.33 rows=20 width=12)
  Node/s: All datanodes
  -> Seq Scan on t1 (cost=16.88..33.08 rows=20 width=12)
      Filter: (hashed SubPlan 1)
      SubPlan 1
        -> Materialize (cost=0.00..16.48 rows=640 width=4)
            -> Streaming(type: BROADCAST) (cost=0.00..16.29 rows=160 width=4)
                Spawn on: All datanodes
                -> Seq Scan on t2 (cost=0.00..16.16 rows=40 width=4)
(9 rows)

```

6.10.16 Hints Specifying Not to Use Global Plan Cache

Function

When global plan cache is enabled, you can use the `no_gpc` hint to force a single query statement not to share the plan cache globally. Only the plan cache within the current session lifecycle is retained.

Syntax

```
no_gpc
```

NOTE

This parameter takes effect only for statements executed by PBE when `enable_global_plancache` is set to `on`.

Example

```

gaussdb=# deallocate all;
DEALLOCATE ALL
gaussdb=# prepare p1 as insert /*+ no_gpc */ into t1 select c1,c2 from t2 where c1=$1;
PREPARE
gaussdb=# execute p1(3);
INSERT 0 1
gaussdb=# select * from db_perf.global_plancache_status where schema_name='public' order by 1,2;
nodename | query | refcount | valid | databaseid | schema_name | params_num | func_id | pkg_id | stmt_id
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)

```

No result exists in the `dbe_perf.global_plancache_status` view, that is, no plan is cached globally.

6.10.17 Hint of Parameterized Paths at the Same Level

Function

The `predpush_same_level` and `nestloop_index` hints are used to specify the generation of parameterized paths between tables or materialized views at the same level.

For details about cross-layer parameterized path hints, see [Parameterized Path Hint](#).

Syntax

```
predpush_same_level([[@queryblock] src, dest)
predpush_same_level([[@queryblock] src1 src2 ..., dest)
[no] nestloop_index([[@queryblock] dest[, index_list]) -- With indexes
[no] nestloop_index([[@queryblock] dest[, (src1 src2 ...)]) -- With tables
```

NOTE

The `predpush_same_level` parameter takes effect only when the `predpushforce` option in `rewrite_rule` is enabled.

`nestloop_index` has no requirement on `rewrite_rule`.

Parameter Description

- **no** indicates that the parameterized path of hints is not used.
- For details about `@queryblock`, see [Hint Specifying the Query Block Where the Hint Is Located](#). `@queryblock` can be omitted, indicating that the hint takes effect in the current query block.
- **dest** is the target table of the parameterized path, that is, the table where the indexes are located.
- **src** is the parameter table of the parameterized path.
- **index_list** is the index sequence used by the parameterized path, which consists of character strings separated by spaces.

Examples

To view the following plan example, you need to set the following parameters:

```
set enable_fast_query_shipping = off;
set enable_stream_operator = on;
```

1. Examples of `nestloop_index`:

- Transfer **t2.c1** and **t3.c2** of **t2** and **t3** to the **t1** table for index scanning (parameterized path).
gaussdb=# explain (costs off) select /*+nestloop_index(t1,(t2 t3)) */* from t1,t2,t3 where t1.c1 = t2.c1 and t1.c2 = t3.c2;

QUERY PLAN

```
-----
Streaming (type: GATHER)
  Node/s: All datanodes
  -> Nested Loop
    -> Streaming (type: BROADCAST)
```

```

Spawn on: All datanodes
-> Seq Scan on t3
-> Nested Loop
-> Seq Scan on t2
-> Index Scan using it1 on t1
    Index Cond: ((c1 = t2.c1) AND (c2 = t3.c2))
(10 rows)

```

- Perform an index scan on **it1** of the **t1** table (parameterized path).

```

gaussdb=# explain (costs off) select /*+NestLoop_Index(t1,it1) */ from t1,t2 where t1.c1 = t2.c1;
QUERY PLAN

```

```

-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Nested Loop
-> Seq Scan on t2
-> Index Scan using it1 on t1
    Index Cond: (c1 = t2.c1)
(6 rows)

```

2. Example of predpush_same_level:

- Prepare parameters.

```

gaussdb=# set rewrite_rule = 'predpushforce';
SET
gaussdb=# set enable_fast_query_shipping=off;
SET

```

- Run the following statement to view the plan:

```

gaussdb=# explain select * from t1, t2 where t1.c1 = t2.c1;
QUERY PLAN

```

```

-----
Streaming (type: GATHER) (cost=16.98..34.22 rows=40 width=24)
Node/s: All datanodes
-> Hash Join (cost=16.36..32.66 rows=40 width=24)
    Hash Cond: (t1.c1 = t2.c1)
-> Seq Scan on t1 (cost=0.00..16.16 rows=40 width=12)
-> Hash (cost=16.16..16.16 rows=40 width=12)
    -> Seq Scan on t2 (cost=0.00..16.16 rows=40 width=12)
(7 rows)

```

- The filter condition **t1.c1 = t2.c2** is displayed on **Join**. In this case, **predpush_same_level(t1, t2)** can be used to push the condition down to the scan operator of **t2**.

```

gaussdb=# explain select /*+predpush_same_level(t1, t2)*/ * from t1, t2 where t1.c1 = t2.c1;
QUERY PLAN

```

```

-----
Streaming (type: GATHER) (cost=0.62..70.20 rows=40 width=24)
Node/s: All datanodes
-> Nested Loop (cost=0.00..68.64 rows=40 width=24)
-> Seq Scan on t1 (cost=0.00..16.16 rows=40 width=12)
-> Index Scan using it2 on t2 (cost=0.00..3.27 rows=1 width=12)
    Index Cond: (c1 = t1.c1)
(6 rows)

```

NOTICE

- You can specify multiple **src** parameters in the same condition.
- If the specified **src** and **dest** conditions do not exist or do not meet the parameterized path requirements, this hint does not take effect.
- If a stream operator exists on the **dest** scanning operator, this hint does not take effect.

6.10.18 Hint for Setting Slow SQL Control Rules

Function

Users can set the execution time, maximum execution time, and maximum IOPS for SQL statements marked as slow SQL statements.

Syntax

```
wlmrule("time_limit,max_execute_time,max_iops")
```

NOTE

This parameter is valid only for SELECT statements executed by non-sysadmin or non-monitoradmin users when **enable_thread_pool** is set to **on**.

- **time_limit**: execution time of an SQL statement marked as a slow statement. The value ranges from **0** to *INT_MAX*. This parameter takes effect on both CNs and DN.
- **max_execute_time**: maximum execution time of an SQL statement. If the execution time exceeds the value of this parameter, the SQL statement is forcibly canceled and exits. The value ranges from **0** to *INT_MAX*. This parameter takes effect only on DN. If the value of **max_execute_time** is less than or equal to the value of **time_limit**, the rule does not take effect.
- **max_iops**: maximum IOPS of an SQL statement marked as a slow SQL statement. This parameter is valid only when **use_workload_manager** is set to **on**. The IOPS limit applies logical I/O control. For details about the definition of IOPS, see the definition of **io_control_unit**. The value can be **Low**, **Medium**, **High**, **None**, or **0-INT_MAX**. This parameter takes effect only on DN.

Example

```
select /*+ wlmrule("100,500,1") */ * from t2 order by b limit 1;
```

It indicates that the execution time of the current statement marked as a slow SQL statement is 100 ms, the maximum execution time is 500 ms, and the maximum IOPS is 1.

6.10.19 Bitmap Scan Hints

Function

These hints generate a bitmap scan path by using the specified index on the target table. The path that meets the hint requirement is selected from the paths that can be generated by the optimizer.

Syntax

```
[no] bitmapscan([@queryblock] table [index_list])
```

Parameter Description

- **no** indicates that the scan of hints is not used.
- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.
- **table** indicates the target table of the bitmap scan.
- **index_list** indicates the index used by the bitmap scan.

Example

```
gaussdb=# explain(costs off) select /*+ BitmapScan(t1 it1 it3)*/ from t1 where (t1.c1 = 5 or t1.c2=6) or (t1.c3=3 or t1.c2=7);
```

QUERY PLAN

```
-----  
Streaming (type: GATHER)  
Node/s: All datanodes  
-> Bitmap Heap Scan on t1  
  Recheck Cond: ((c1 = 5) OR (c2 = 6) OR (c3 = 3) OR (c2 = 7))  
  -> BitmapOr  
    -> Bitmap Index Scan on it1  
      Index Cond: (c1 = 5)  
    -> Bitmap Index Scan on it3  
      Index Cond: (c2 = 6)  
    -> Bitmap Index Scan on it3  
      Index Cond: (c3 = 3)  
    -> Bitmap Index Scan on it3  
      Index Cond: (c2 = 7)  
(13 rows)
```

NOTE

The path that meets the bitmap scan hint is selected from the existing index paths. Because the index path construction space is large and the optimizer prunes the paths, if any index path is not generated, the path cannot be constructed.

6.10.20 Hint for Inner Table Materialization During Join

Function

These hints materialize inner tables when specifying the inner tables to be joined.

Syntax

```
[no] materialize_inner([@queryblock] inner_table_list)
```

Parameter Description

- **no** indicates that the materialization of hints is not used.
- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.
- **inner_table_list**: a list of inner tables to be materialized during the join operation. The value is a character string separated by spaces.

Example

Table **t1** is an inner table to be materialized, and the result of (t1 t2) is materialized as a joined inner table.

```
gaussdb=# explain (costs off) select /*+materialize_inner(t1) materialize_inner(t1 t2)*/ * from t1,t2,t3  
where t1.c3 = t2.c3 and t2.c2=t3.c2 and t1.c2=5;
```

```
QUERY PLAN  
-----  
Streaming (type: GATHER)  
Node/s: All datanodes  
-> Nested Loop  
Join Filter: (t2.c2 = t3.c2)  
-> Seq Scan on t3  
-> Materialize  
-> Streaming(type: BROADCAST)  
Spawn on: All datanodes  
-> Nested Loop  
Join Filter: (t1.c3 = t2.c3)  
-> Seq Scan on t2  
-> Materialize  
-> Streaming(type: BROADCAST)  
Spawn on: All datanodes  
-> Index Scan using it3 on t1  
Index Cond: (c2 = 5)  
  
(16 rows)
```

6.10.21 AGG Hint

Function

You can specify the AGG method when performing the AGG algorithm.

Syntax

```
use_hash_agg[(@queryblock)], use_sort_agg[(@queryblock)]
```

Parameter Description

- For details about **@queryblock**, see [Hint Specifying the Query Block Where the Hint Is Located](#). **@queryblock** can be omitted, indicating that the hint takes effect in the current query block. If it is not specified, the hint does not have parentheses ().

Example

1. Use hash aggregation.

```
gaussdb=# explain (costs off) select c1 from t2 where c1 in( select /*+ use_hash_agg */ t1.c1 from  
t1,t3 where t1.c1=t3.c1 group by 1);
```

```
QUERY PLAN  
-----  
Streaming (type: GATHER)  
Node/s: All datanodes  
-> Hash Join  
Hash Cond: (t2.c1 = t1.c1)  
-> Seq Scan on t2  
-> Hash  
-> HashAggregate  
Group By Key: t1.c1  
-> Hash Join  
Hash Cond: (t1.c1 = t3.c1)  
-> Seq Scan on t1  
-> Hash
```

```

-> Seq Scan on t3
(13 rows)
2. Use use_sort_agg for aggregation and then perform merge join.
gaussdb=# explain (costs off) select c1 from t2 where c1 in( select /*+ use_sort_agg */ t1.c1 from t1,t3
where t1.c1=t3.c1 group by 1);
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Join
Hash Cond: (t2.c1 = t1.c1)
-> Seq Scan on t2
-> Hash
-> Group
Group By Key: t1.c1
-> Sort
Sort Key: t1.c1
-> Hash Join
Hash Cond: (t1.c1 = t3.c1)
-> Seq Scan on t1
-> Hash
-> Seq Scan on t3
(15 rows)

```

6.11 Checking the Implicit Conversion Performance

In some scenarios, implicit data type conversion may cause performance problems. For example:

```

SET enable_fast_query_shipping = off;
CREATE TABLE t1(c1 VARCHAR, c2 VARCHAR);
CREATE INDEX on t1(c1);
EXPLAIN verbose SELECT * FROM t1 WHERE c1 = 10;

```

The execution plan of the preceding query is as follows.

```

-----
QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.06..13.29 rows=1 width=64)
Output: c1, c2
Node/s: All datanodes
-> Seq Scan on public.t1 (cost=0.00..13.20 rows=1 width=64)
Output: c1, c2
Distribute Key: c1
Filter: ((t1.c1)::bigint = 10)
(7 rows)

```

The data type of **c1** is **varchar**. When the filter criterion is **c1 = 10**, the optimizer implicitly converts the data type of **c1** to **bigint** by default. As a result, the following two consequences occur:

- DN tailoring is not allowed. The plan is delivered to all DNs for execution.
- The Index Scan mode cannot be used to scan data in the plan.

These may cause performance problems.

After knowing the causes, you can rewrite the SQL statements. In the preceding scenario, you only need to convert the constant display in the filter criteria to the varchar type. The result is as follows:

```

EXPLAIN verbose SELECT * FROM t1 WHERE c1 = 10::varchar;

```

```
-----  
QUERY PLAN  
-----  
Streaming (type: GATHER) (cost=0.06..8.36 rows=1 width=64)  
  Output: c1, c2  
  Node/s: datanode2  
  -> Index Scan using t1_c1_idx on public.t1 (cost=0.00..8.27 rows=1 width=64)  
      Output: c1, c2  
      Distribute Key: c1  
      Index Cond: ((t1.c1)::text = '10'::text)  
(7 rows)
```

To identify the performance impact of implicit type conversion in advance, you can use the GUC parameter **check_implicit_conversions**. After this parameter is enabled, the system checks the index columns that are implicitly converted in the query in the path generation phase. If no candidate index scan path is generated for the index columns, an error message is displayed. For example:

```
SET check_implicit_conversions = on;  
SELECT * FROM t1 WHERE c1 = 10;  
ERROR: There is no optional index path for index column: "t1"."c1".
```

NOTE

- The **check_implicit_conversions** parameter is used only to check for potential performance problems caused by implicit type conversion. In the formal production environment, set this parameter to **off** (default value) to disable it.
- When enabling **check_implicit_conversions**, you must disable **enable_fast_query_shipping**. Otherwise, you cannot view the result of restoring the implicit type conversion.
- A candidate path of a table may include multiple possible data scan modes such as sequential scan and index scan. A table scan mode used in the final execution plan is determined by the cost of the execution plan. Therefore, even if a candidate path for index scan is generated, other scan modes may also be used in the final execution plan.

6.12 Tuning with SQL PATCH

SQL PATCH is designed for database administrators (DBAs), O&M personnel, and other roles who need to optimize SQL statements. If performance problems caused by poor plans of service statements are identified through other O&M views or fault locating methods, you can create an SQL patch to optimize service statements based on hints. Currently, the following hints are supported: number of rows, scanning mode, join mode, join sequence, PBE custom/generic plan selection, statement-level parameter setting, and parameterized path. In addition, in case that services are unavailable due to internal system errors that are triggered by specific statements, you can create SQL patches to rectify single-point failures without changing service statements. In this way, errors can be reported in advance to avoid greater loss.

Constraints

1. Patches can be created only by unique SQL ID. If unique SQL IDs conflict, SQL patches that are used for hint-based optimization may affect performance but do not affect semantic correctness.
2. Only hints that do not change SQL semantics can be used as patches. SQL rewriting is not supported.

3. This tool is not applicable to logical backup and restoration.
4. SQL patches cannot be created on DNs.
5. Only the initial user, O&M administrator, monitoring administrator, and system administrator have the permission to perform this operation.
6. Patches are not shared between databases. When creating SQL patches, you need to connect to the target database. If the CN where the SQL PATCH is created is removed and a full build is triggered, the SQL PATCH in the target CN of the full build is inherited. Therefore, you are advised to create the corresponding SQL PATCH on each CN.
7. CNs do not share SQL patches because their unique SQL IDs are different. You need to manually create SQL patches on different CNs.
8. SQL patches in a stored procedure and global SQL patches cannot coexist.
9. SQL patches cannot be used for precompiled statements that are executed using the PREPARE + EXECUTE syntax.
10. It is not recommended that the SQL patches be used in the database for a long time. It should be used only as a workaround. If the database service is unavailable due to a kernel fault triggered by a specific statement or SQL hints are used for performance tuning, you must rectify the service fault or upgrade the kernel as soon as possible. After the upgrade, the method of generating unique SQL IDs may change. Therefore, the workaround may become invalid.
11. Currently, except DML statements, unique SQL IDs of SQL statements (such as CREATE TABLE) are generated by hashing the statement text. Therefore, SQL PATCH is sensitive to uppercase and lowercase letters, spaces, and line breaks. That is, even statements of different texts have the same semantics, you still need to create different SQL patches for them. For DML operations, SQL PATCH can take effect for the same statement with different input parameters, regardless of uppercase letters, lowercase letters, and spaces.

Example

The SQL patch is implemented based on the unique SQL ID. Therefore, you need to enable related O&M parameters (**enable_resource_track = on**, **instr_unique_sql_count > 0**) for the SQL patch to take effect. The unique SQL ID can be obtained from both the WDR and slow SQL view. You need to specify the unique SQL ID when creating the SQL patch. For SQL statements in a stored procedure, you need to set **instr_unique_sql_track_type** to **'all'** and query unique SQL ID in the `db_perf.statement_history` view.

The following provides a simple example.

Scenario 1: Use SQL PATCH to optimize specific statements based on hints.

```
gaussdb=# create table hint_t1(a int, b int, c int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# create index on hint_t1(a);
CREATE INDEX
gaussdb=# insert into hint_t1 values(1,1,1);
INSERT 0 1
gaussdb=# analyze hint_t1;
ANALYZE
gaussdb=# set track_stmt_stat_level = 'L1,L1'; -- Enable full SQL statistics.
SET
```

```
gaussdb=# set enable_fast_query_shipping = off; -- Disable statement pushdown so that plans are
generated on the CN.
SET
gaussdb=# set explain_perf_mode = normal; --Adjust the plan display format.
SET
gaussdb=# select * from hint_t1 where hint_t1.a = 1; -- Execute SQL statements.
 a | b | c
---+---+---
 1 | 1 | 1
(1 row)
gaussdb=# \x --Switch to the extended display mode to facilitate plan observation.
Expanded display is on.
gaussdb=# select unique_query_id, query, query_plan from db_perf.statement_history where query like
'%hint_t1%';-- Obtain the query plan and unique SQL ID. This statement needs to query the slow SQL view
db_perf.statement_history in the postgres database.
-[ RECORD 1 ]---+-----
unique_query_id | 3929365485
query           | select * from hint_t1 where hint_t1.a = ?;
query_plan      | Coordinator Name: coordinator1
                | Streaming (type: GATHER) (cost=0.06..1.11 rows=1 width=12)
                | Node/s: datanode1
                | -> Seq Scan on hint_t1 (cost=0.00..1.01 rows=1 width=12)
                |     Filter: (a = '****')
                |
gaussdb=# \x --Disable the extended display mode.

gaussdb=# select * from db_sql_util.create_hint_sql_patch('patch1', 3929365485, 'indexscan(hint_t1)');
create_hint_sql_patch
-----
 t
(1 row)
gaussdb=# set track_stmt_stat_level = 'L1,L1'; --Reset parameters after the switching.
SET
gaussdb=# set enable_fast_query_shipping = off;
SET
gaussdb=# explain select * from hint_t1 where hint_t1.a = 1;
NOTICE: Plan influenced by SQL hint patch
          QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.06..8.36 rows=1 width=12)
Node/s: datanode1
-> Index Scan using hint_t1_a_idx on hint_t1 (cost=0.00..8.27 rows=1 width=12)
    Index Cond: (a = 1)
(4 rows)

gaussdb=# select * from hint_t1 where hint_t1.a = 1; -- Run the statement again.
 a | b | c
---+---+---
 1 | 1 | 1
(1 row)

gaussdb=# \x
Expanded display is on.

gaussdb=# select unique_query_id, query, query_plan from db_perf.statement_history where query like
'%hint_t1%'; -- The query plan has been changed.
-[ RECORD 1 ]---+-----
unique_query_id | 3929365485
query           | select * from hint_t1 where hint_t1.a = ?;
query_plan      | Coordinator Name: coordinator1
                | Streaming (type: GATHER) (cost=0.06..1.11 rows=1 width=12)
                | Node/s: datanode1
                | -> Seq Scan on hint_t1 (cost=0.00..1.01 rows=1 width=12)
                |     Filter: (a = '****')
                |
-[ RECORD 2 ]---+-----
unique_query_id | 3929365485
```

```

query      | select * from hint_t1 where hint_t1.a = ?;
query_plan | Coordinator Name: coordinator1
           | Streaming (type: GATHER) (cost=0.06..8.36 rows=1 width=12)
           | Node/s: datanode1
           | -> Index Scan using hint_t1_a_idx on hint_t1 (cost=0.00..8.27 rows=1 width=12)
           |      Index Cond: (a = '****')

```

Scenario 2: Run the SQL PATCH command to report an error for a specific statement in advance.

```

gaussdb=# select * from db_sql_util.drop_sql_patch('patch1'); -- Delete patch 1.
drop_sql_patch
-----
t
(1 row)
gaussdb=# select * from db_sql_util.create_abort_sql_patch('patch2', 3929365485); -- Create an abort
patch for the unique SQL ID of the statement.
create_abort_sql_patch
-----
t
(1 row)

gaussdb=# select * from hint_t1 t1 where t1.a = 1; -- An error is reported in advance when the statement is
executed again.
ERROR: Statement 2578396627 canceled by abort patch patch2

```

Scenario 3: Create an SQL patch for SQL statements in a stored procedure.

```

gaussdb=# create table test_proc_patch(a int,b int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# insert into test_proc_patch values(1,2);
INSERT 0 1
gaussdb=# create procedure mypro() as num int;
gaussdb$# begin
gaussdb$# select b into num from test_proc_patch where a = 1;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# set track_stmt_stat_level = 'L0,L1';
SET
gaussdb=# select b from test_proc_patch where a = 1;
 b
---
 2
(1 row)

gaussdb=# call mypro();
 mypro
-----
(1 row)

gaussdb=# select unique_query_id, query, query_plan, parent_unique_sql_id from
db_perf.statement_history where query like '%call mypro();%' or query like '%test_proc_patch%';
unique_query_id |          query          |          query_plan          |          parent_unique_sql_id
-----+-----+-----+-----
+-----+-----+-----+-----
2859505004 | select b from test_proc_patch where a = ?; | | |
|          0          | | | |
+| 2502737203 | call mypro();          | Coordinator Name: cn1      | |
|          0          | | | | |
|          |          | Function Scan on mypro (cost=0.25..0.26 rows=1 width=4)+ | |
|          |          |          |          |          |
2859505004 | select b from test_proc_patch where a = ?; | Coordinator Name:

```

```

cn1          +|          2502737203
          |          | Data Node Scan (cost=0.00..0.00 rows=0 width=0)  +|
          |          | Node/s: datanode1          +|
          |          |
(3 rows)

gaussdb=# select * from dbe_sql_util.create_abort_sql_patch('patch1',2859505004,2502737203); -- Restrict
that the abort patch takes effect only for statements in the stored procedure.
create_abort_sql_patch
-----
t
(1 row)

gaussdb=# select patch_name,unique_sql_id,parent_unique_sql_id,enable,abort,hint_string from
gs_sql_patch where patch_name = 'patch1'; -- Check whether the patch is correctly created and takes effect.
patch_name | unique_sql_id | parent_unique_sql_id | enable | abort | hint_string
-----+-----+-----+-----+-----+-----
patch1    | 2859505004    | 2502737203          | t      | t      |
(1 row)

gaussdb=# select b from test_proc_patch where a = 1;
b
----
2
(1 row)

gaussdb=# call mypro();
ERROR: Statement 2859505004 canceled by abort patch patch1
CONTEXT: SQL statement "select b          from test_proc_patch where a = 1"
PL/SQL function mypro() line 3 at SQL statement
    
```

Scenario 4: Install the SQL patch for the same slow SQL statement on each CN.

```

-- Find the slow SQL statement and plan on each node. (This function requires the monadmin permission.)
select node_name, unique_query_id, start_time, query, query_plan from
dbe_perf.get_global_full_sql_by_timestamp(<start_time>, <end_time>);

-- Observe and analyze the returned slow SQL statement and plan, and perform local optimization and
verification to obtain a proper hint_str.

-- Run the following statement on any CN to create an SQL patch. node_name and unique_query_id are
obtained from step 1.
select * from dbe_sql_util.create_remote_hint_sql_patch(<node_name>, <patch_name>, <unique_query_id>,
<hint_str>);
    
```

Helpful Links

The following table lists the system functions, system catalogs, system views, and interface functions related to SQL PATCH.

Table 6-5 System functions, system catalogs, system views, and interface functions related to SQL PATCH

Category	Name	Description
System function	global_sql_patch_func()	SQL patch information on each global node, which is used to return the result of the global_sql_patch view.
System catalog	GS_SQL_PATCH	GS_SQL_PATCH records the status information about all SQL patches.

Category	Name	Description
System view	GLOBAL_SQL_PATCH	GLOBAL_SQL_PATCH stores information about all SQL patches. This view is available only in the PG_CATALOG schema.
Interface function DBE_SQL_UTIL Schema	DBE_SQL_UTIL.create_hint_sql_patch	create_hint_sql_patch creates hint SQL patches on the connected CN and returns whether the execution is successful.
	DBE_SQL_UTIL.create_abort_sql_patch	create_abort_sql_patch creates abort SQL patches on the connected CN and returns whether the execution is successful.
	DBE_SQL_UTIL.drop_sql_patch	drop_sql_patch deletes SQL patches from the connected CN and returns whether the execution is successful.
	DBE_SQL_UTIL.enable_sql_patch	enable_sql_patch enables SQL patches on the connected CN and returns whether the execution is successful.
	DBE_SQL_UTIL.disable_sql_patch	disable_sql_patch disables SQL patches on the connected CN and returns whether the execution is successful.
	DBE_SQL_UTIL.show_sql_patch	show_sql_patch displays the SQL patch corresponding to a specified patch name and returns the running result.
	DBE_SQL_UTIL.create_hint_sql_patch	create_hint_sql_patch creates hint SQL patches and returns whether the execution is successful. This function is an overloaded function of the original function. The value of parent_unique_sql_id can be used to limit the effective range of the hint patch.
	DBE_SQL_UTIL.create_abort_sql_patch	create_abort_sql_patch creates abort SQL patches and returns whether the execution is successful. This function is an overloaded function of the original function. The value of parent_unique_sql_id can be used to limit the effective range of the abort patch.

Category	Name	Description
	DBE_SQL_UTIL.create_remote_hint_sql_patch	create_remote_hint_sql_patch creates hint SQL patches on a specified CN and returns whether the execution is successful.
	DBE_SQL_UTIL.create_remote_abort_sql_patch	create_remote_abort_sql_patch creates abort SQL patches on a specified CN and returns whether the execution is successful.
	DBE_SQL_UTIL.drop_remote_sql_patch	drop_remote_sql_patch deletes SQL patches from a specified CN and returns whether the execution is successful.
	DBE_SQL_UTIL.enable_remote_sql_patch	enable_remote_sql_patch enables SQL patches on a specified CN and returns whether the execution is successful.
	DBE_SQL_UTIL.disable_remote_sql_patch	disable_remote_sql_patch disables SQL patches on a specified CN and returns whether the execution is successful.

6.13 Optimization Cases

6.13.1 Case: Selecting an Appropriate Distribution Key

Symptom

Tables are defined as follows:

```
CREATE TABLE t1 (a int, b int);
CREATE TABLE t2 (a int, b int);
```

The following query is executed:

```
SELECT * FROM t1, t2 WHERE t1.a = t2.b;
```

Optimization Analysis

If **a** is the distribution key of **t1** and **t2**:

```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH (a);
CREATE TABLE t2 (a int, b int) DISTRIBUTE BY HASH (a);
```

Then **Streaming** exists in the execution plan and the data volume is heavy among DNs, as shown in [Figure 6-8](#).

Figure 6-8 Selecting an appropriate distribution key (1)

```
openGauss=> explain select * from t1, t2 where t1.a = t2.b;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=245.40..582.15 rows=240 width=16)
Node/s: All datanodes
-> Hash Join (cost=10.22..24.26 rows=10 width=16)
    Hash Cond: (t1.a = t2.b)
    -> Seq Scan on t1 (cost=0.00..10.10 rows=10 width=8)
    -> Hash (cost=3.79..3.79 rows=10 width=8)
        -> Streaming(type: REDISTRIBUTE) (cost=0.00..3.79 rows=10 width=8)
            Spawn on: All datanodes
            -> Seq Scan on t2 (cost=0.00..10.10 rows=10 width=8)
(9 rows)
```

If **a** is the distribution key of **t1** and **b** is the distribution key of **t2**:

```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH (a);
CREATE TABLE t2 (a int, b int) DISTRIBUTE BY HASH (b);
```

Then **Streaming** does not exist in the execution plan, and the data volume among DN is decreasing and the query performance is increasing, as shown in **Figure 6-9**.

Figure 6-9 Selecting an appropriate distribution key (2)

```
openGauss=> explain select * from t1, t2 where t1.a = t2.b;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=245.40..491.10 rows=240 width=16)
Node/s: All datanodes
-> Hash Join (cost=10.22..20.46 rows=10 width=16)
    Hash Cond: (t1.a = t2.b)
    -> Seq Scan on t1 (cost=0.00..10.10 rows=10 width=8)
    -> Hash (cost=10.10..10.10 rows=10 width=8)
        -> Seq Scan on t2 (cost=0.00..10.10 rows=10 width=8)
(7 rows)
```

6.13.2 Case: Creating an Appropriate Index

Symptom

Query the information about all personnel in the sales department.

```
-- Create a table.
CREATE TABLE staffs (staff_id NUMBER(6) NOT NULL, first_name VARCHAR2(20), last_name
VARCHAR2(25), employment_id VARCHAR2(10), section_id NUMBER(4), state_name VARCHAR2(10), city
VARCHAR2(10));
CREATE TABLE sections (section_id NUMBER(4), place_id NUMBER(4), section_name VARCHAR2(20));
CREATE TABLE states (state_id NUMBER(4));
CREATE TABLE places (place_id NUMBER(4), state_id NUMBER(4));
-- Query before optimization.
EXPLAIN SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
-- Create indexes.
CREATE INDEX loc_id_pk ON places(place_id);
CREATE INDEX state_c_id_pk ON states(state_id);
-- Query after optimization.
```

```
EXPLAIN SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
```

Optimization Analysis

The original execution plan is as follows before creating the **places.place_id** and **states.state_id** indexes:

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	2	254	54.08
2	-> Sort	2	254	53.90
3	-> Nested Loop (4,5)	2	254	53.88
4	-> Seq Scan on staffs	20	266	13.13
5	-> Materialize	4	12	40.37
6	-> Streaming(type: BROADCAST)	4	12	40.36
7	-> Nested Loop (8,9)	2	12	40.20
8	-> Seq Scan on states	20	12	13.13
9	-> Materialize	2	24	26.69
10	-> Streaming(type: REDISTRIBUTE)	2	24	26.68
11	-> Nested Loop (12,14)	2	24	26.57
12	-> Streaming(type: REDISTRIBUTE)	1	24	13.28
13	-> Seq Scan on sections	1	24	13.16
14	-> Seq Scan on places	20	24	13.13

(14 rows)

Predicate Information (identified by plan id)

```
3 --Nested Loop (4,5)
  Join Filter: (sections.section_id = staffs.section_id)
7 --Nested Loop (8,9)
  Join Filter: (places.state_id = states.state_id)
11 --Nested Loop (12,14)
  Join Filter: (sections.place_id = places.place_id)
13 --Seq Scan on sections
  Filter: ((section_name)::text = 'Sales'::text)
```

(8 rows)

The optimized execution plan is as follows (two indexes have been created on the **places.place_id** and **states.state_id** columns in [Symptom](#)):

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	2	254	42.26
2	-> Sort	2	254	42.08
3	-> Nested Loop (4,5)	2	254	42.06
4	-> Seq Scan on staffs	20	266	13.13
5	-> Materialize	4	12	28.55
6	-> Streaming(type: BROADCAST)	4	12	28.54
7	-> Nested Loop (8,13)	2	12	28.38
8	-> Streaming(type: REDISTRIBUTE)	2	24	21.66
9	-> Nested Loop (10,12)	2	24	21.56
10	-> Streaming(type: REDISTRIBUTE)	1	24	13.28
11	-> Seq Scan on sections	1	24	13.16
12	-> Index Scan using loc_id_pk on places	1	24	8.27
13	-> Index Only Scan using state_c_id_pk on states	1	12	3.35

(13 rows)

Predicate Information (identified by plan id)

```
3 --Nested Loop (4,5)
  Join Filter: (sections.section_id = staffs.section_id)
11 --Seq Scan on sections
```

```

Filter: ((section_name)::text = 'Sales'::text)
12 --Index Scan using loc_id_pk on places
    Index Cond: (place_id = sections.place_id)
13 --Index Only Scan using state_c_id_pk on states
    Index Cond: (state_id = places.state_id)
(8 rows)
    
```

6.13.3 Case: Adjusting Distribution Keys

Symptom

During a site test, the information is displayed after **EXPLAIN ANALYZE** is run:

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	Streaming (type: GATHER)	94138.404	0	670912	292KB			73	102576573.63
2	Insert on temp_calc_emprate0101 t3	[93259.538,93430.438]	310	670912	[1108KB, 1108KB]	1MB		73	102534641.63
3	Streaming (type: REDISTRIBUTE)	[93259.507,93430.400]	310	670912	[2091KB, 2093KB]	1MB		73	102534641.63
4	Subquery Scan on "SELECT"	[93212.430,93419.986]	310	670912	[7KB, 7KB]	1MB		73	102533776.78
5	HashAggregate	[93212.425,93419.980]	310	670912	[145KB, 197KB]	10MB	[65,65]	45	102533645.74
6	Streaming (type: REDISTRIBUTE)	[93212.374,93419.924]	586	670934	[2091KB, 2093KB]	1MB		45	102533305.05
7	Hash Join (8,12)	[2657.406,93339.924]	586	670934	[20KB, 20KB]	1MB		45	102532655.39
8	Seq Scan on s_riskrate_setting a	[38,885,2940,983]	77282027	78594218	[812KB, 903KB]	1MB		56	275264.71
9	Hash	[1241.418,2713.381]	8536241	8536241	[1031KB, 97803KB]	10MB	[48,48]	46	50970.88
10	Streaming (type: REDISTRIBUTE)	[210.226,2437.195]	8536241	8536241	[2091KB, 2093KB]	1MB		46	50970.88
11	Seq Scan on temp_calc_emprate0101 b	[86.790,141.293]	8536241	8536241	[16KB, 16KB]	1MB		46	11564.79

According to the execution information, Hash Join becomes the performance bottleneck of the whole plan. Based on the execution time of hash join ranging from 2657.406 to 93339.924 (for details about the values, see [Execution Plan Details](#)), it can be seen that severe skew occurs on different DN during the Hash Join operation.

In the memory information (as shown in the following figure), it can be seen that the data skew occurs in the memory usage of each node.

```

----- Memory Information (identified by plan id) -----
Coordinator:
--Query Peak Memory: 4MB
Datanode:
--Max Query Peak Memory: 118MB
--Min Query Peak Memory: 24MB
--12 --Hash
-----Max Buckets: 131072  Max Batches: 1  Max Memory Usage: 91857kB
-----Min Buckets: 131072  Min Batches: 1  Min Memory Usage: 0kB
(8 rows)
    
```

Optimization Analysis

The preceding two features indicate that this SQL statement has extremely serious computing unbalance. The further lower-layer analysis on the Hash Join operator shows that serious computing skew **[38.885,2940.983]** occurs in **Seq Scan on s_riskrate_setting**. Based on the description of the Scan, we can infer that the performance problems of this plan lie in data skew occurred in the **s_riskrate_setting** table. Later, it is proved that serious data skew occurred in the **s_riskrate_setting** table. After performance optimization, the execution time is reduced from 94s to 50s.

6.13.4 Case: Adjusting the GUC Parameter best_agg_plan

Symptom

The **agg_t1** table is defined as follows:

```
create table agg_t1(a int, b int, c int) distribute by hash(a);
```

Assume that the distribution key of the result set provided by the agg lower-layer operator is **setA**, and the **group by** column of the agg operation is **setB**, the agg operations can be performed in two scenarios in the Stream framework.

1. **setA** is a subset of **setB**.

In this scenario, the aggregation result of the lower-layer is correct and can be directly used by upper-level operators. Example:

```
gaussdb=# explain select a, count(1) from agg_t1 group by a;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 20 | 12 | 14.23
2 | -> HashAggregate | 20 | 12 | 13.30
3 | -> Seq Scan on agg_t1 | 20 | 4 | 13.13
(3 rows)
```

2. **setA** is not a subset of **setB**.

In this scenario, the Stream execution framework is classified into the following three plans:

- hashagg + gather(redistribute) + hashagg;
- redistribute + hashagg(+gather);
- hashagg + redistribute + hashagg(+gather).

GaussDB provides the GUC parameter **best_agg_plan** to intervene the execution plan, and forces the plan to generate the corresponding execution plan. This parameter can be set to **0, 1, 2, 3**.

- When the parameter is set to **1**, the first plan is forcibly generated.
- When the parameter is set to **2** and if the **group by** column can be redistributed, the second plan is forcibly generated. Otherwise, the first plan is generated.
- When the parameter is set to **3** and if the **group by** column can be redistributed, the third plan is generated. Otherwise, the first plan is generated.
- When the parameter is set to **0**, the query optimizer chooses the most optimal plan by the three preceding plans' evaluation cost.

For details, see the following figure.

```
gaussdb=# set best_agg_plan to 1;
SET
gaussdb=# explain select b,count(1) from t1 group by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> HashAggregate | 8 | 4 | 15.83
2 | -> Streaming (type: GATHER) | 25 | 4 | 15.83
3 | -> HashAggregate | 25 | 4 | 14.33
4 | -> Seq Scan on t1 | 30 | 4 | 14.14
(4 rows)
gaussdb=# set best_agg_plan to 2;
SET
gaussdb=# explain select b,count(1) from t1 group by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 4 | 15.85
2 | -> HashAggregate | 30 | 4 | 14.60
3 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 14.45
4 | -> Seq Scan on t1 | 30 | 4 | 14.14
(4 rows)
gaussdb=# set best_agg_plan to 3;
SET
gaussdb=# explain select b,count(1) from t1 group by b;
id | operation | E-rows | E-width | E-costs
```

```

-----+-----+-----+-----+
 1 | -> Streaming (type: GATHER)          | 30 | 4 | 15.84
 2 | -> HashAggregate                    | 30 | 4 | 14.59
 3 | -> Streaming(type: REDISTRIBUTE)    | 25 | 4 | 14.59
 4 | -> HashAggregate                    | 25 | 4 | 14.33
 5 | -> Seq Scan on t1                   | 30 | 4 | 14.14
(5 rows)

```

Optimization

Generally, the optimizer chooses an optimal execution plan, but the cost estimation, especially that of the intermediate result set, has large deviations, which may result in large deviations in agg calculation. In this case, you need to use **best_agg_plan** to adjust the agg calculation model.

When the aggregation convergence ratio is very small, that is, the number of result sets does not become small obviously after the agg operation (5 times is a critical point), you can select the redistribute+hashagg or hashagg+redistribute+hashagg execution mode.

6.13.5 Case: Rewriting SQL Statements to Eliminate Subqueries

Symptom

```

select
  1,
  (select count(*) from customer_address_001 a4 where a4.ca_address_sk = a.ca_address_sk) as GZCS
from customer_address_001 a;

```

This SQL performance is poor. SubPlan exists in the execution plan as follows:

```

openGauss=# explain select 1,(select count(*)
openGauss(#          from customer_address_001 a4
openGauss(#          where a4.ca_address_sk = a.ca_address_sk
openGauss(#          ) as GZCS from customer_address_001 a;
 id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+
 1 | -> Streaming (type: GATHER)          | 320 | 4 | 4529.27
 2 | -> Seq Scan on customer_address_001 a | 320 | 4 | 4496.27
 3 | -> Aggregate [2, SubPlan 1]         | 32 | 4 | 139.50
 4 | -> Result                            | 10240 | 4 | 138.69
 5 | -> Materialize                       | 10240 | 4 | 138.69
 6 | -> Streaming(type: BROADCAST)       | 10240 | 4 | 137.09
 7 | -> Seq Scan on customer_address_001 a4 | 320 | 4 | 32.32
(7 rows)

```

Optimization

The core of this optimization is to eliminate subqueries. Based on the service scenario analysis, **a.ca_address_sk** is not null. In terms of SQL syntax, you can rewrite the SQL statement as follows:

```

select
  count(*)
from customer_address_001 a4, customer_address_001 a
where a4.ca_address_sk = a.ca_address_sk
group by a.ca_address_sk;

```

 NOTE

To ensure that the modified statements have the same functions, **not null** is added to **customer_address_001.ca_address_sk**.

6.13.6 Case: Rewriting SQL Statements to Eliminate Pruning Interference

Symptom

In a test at a site, **ddw_f10_op_cust_asset_mon** is a partitioned table and the partitioning key is **year_mth** whose value is a combined string of month and year values.

The following figure shows the tested SQL statements:

```
select count(1) from t_ddw_f10_op_cust_asset_mon b1 where b1.year_mth between
to_char(add_months(to_date('20170222','yyyymmdd'), -11), 'yyyymm') and substr('20170222', 1, 6);
```

The test result shows the Scan operation on the tables in the SQL statement takes 135s. This may be the performance bottleneck.

 NOTE

add_months is a local adaptation function.

```
CREATE OR REPLACE FUNCTION ADD_MONTHS(date, integer) RETURNS date AS $$ SELECT
CASE WHEN (EXTRACT(day FROM $1) = EXTRACT(day FROM (date_trunc('month', $1) + INTERVAL
'1 month - 1 day')) THEN date_trunc('month', $1) + CAST($2 + 1 || ' month - 1 day' as interval)
ELSE $1 + CAST($2 || ' month' as interval) END $$ LANGUAGE SQL IMMUTABLE;
```

Optimization

According to the statement execution plan, the base table filter is displayed as follows:

```
Filter: (((year_mth)::text <= '201702'::text) AND ((year_mth)::text >=
to_char(add_months(to_date('20170222'::text, 'YYYYMMDD'::text), (-11)), 'YYYYMM'::text)))
```

The query condition expression `to_char(add_months(to_date('20170222','yyyymmdd'),-11),'yyyymm')` exists in the filter condition, and this non-constant expression cannot be used for pruning. Therefore, all data of query statements in the partitioned tables is scanned.

`to_date` and `to_char` are stable functions as queried in `pg_proc`. According to the function behavior described in the database, this type of functions cannot be converted to Const values in the preprocessing phase, which is the root cause why partition pruning cannot be performed.

Based on the preceding analysis, the optimization expression can be used for partition pruning, which is the key to performance optimization. The original SQL statements can be written to as follows:

```
select count(1) from t_ddw_f10_op_cust_asset_mon b1 where b1.year_mth
between(substr(ADD_MONTHS('20170222'::date, -11), 1, 4)||substr(ADD_MONTHS('20170222'::date, -11),
6, 2)) and substr('20170222', 1, 6);
```

The execution time of modified SQL statements is reduced from 135s to 18s.

6.13.7 Case: Rewriting SQL Statements and Deleting in-clause

Symptom

in-clause/any-clause is a common SQL statement constraint. Sometimes, the clause following **in** or **any** is a constant. For example:

```
select count(1) from calc_empfyc_c1_result_tmp_t1 where ls_pid_cusr1 in ('20120405', '20130405');
```

Or

```
select count(1) from calc_empfyc_c1_result_tmp_t1 where ls_pid_cusr1 in any('20120405', '20130405');
```

Sometimes, the **in** or **any** clause is used as follows:

```
SELECT ls_pid_cusr1,COALESCE(max(round((current_date-bthdate)/365)),0)FROM  
calc_empfyc_c1_result_tmp_t1 t1,p10_md_tmp_t2 t2WHERE t1.ls_pid_cusr1 = any(values(id),(id15))GROUP  
BY ls_pid_cusr1;
```

id and **id15** are columns in **p10_md_tmp_t2**, and **t1.ls_pid_cusr1 = any(values(id),(id15))** is equivalent to **t1.ls_pid_cusr1 = id** or **t1.ls_pid_cusr1 = id15**.

Therefore, join-condition is essentially an inequality, and nestloop must be used for this unequal join operation. The corresponding execution plan is as follows.

```
Streaming (type: GATHER) (cost=1641429284.14..1641429283.98 rows=3840 width=49)
Node/s: All DataNodes
-> Insert on channel.calc_empfyc_c1_result_tmp (cost=1641429280.14..1641429283.98 rows=3840 width=49)
-> HashAggregate (cost=1641429280.14..1641429283.98 rows=3840 width=25)
   Output: t1.ls_pid_cusr1, COALESCE(max(max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365)::double precision))::numeric, 0))), 0)::numeric)
   Group By Key: t1.ls_pid_cusr1
   -> Streaming (type: REDISTRIBUTE) (cost=820714640.07..820714642.69 rows=3968 width=25)
       Output: t1.ls_pid_cusr1, max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365)::double precision))::numeric, 0))
       Distribute Key: t1.ls_pid_cusr1
       Spwn on: All DataNodes
       -> HashAggregate (cost=820714640.07..820714642.69 rows=3968 width=25)
           Output: t1.ls_pid_cusr1, max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365)::double precision))::numeric, 0))
           Group By Key: t1.ls_pid_cusr1
           -> Nested Loop (cost=0.00..615567760.93 rows=875293350960 width=25)
               Output: t1.ls_pid_cusr1, t2.bthdate
               Join Filter: (SubPlan 1)
               -> Seq Scan on channel.p10_md_tmp_t2 t2 (cost=0.00..127030.52 rows=448523360 width=64)
                   Output: t2.id, t2.id15, t2.bthdate, t2.name
               -> Materialize (cost=0.00..147.29 rows=282608 width=17)
                   Output: t1.ls_pid_cusr1
                   -> Streaming (type: BROADCAST) (cost=0.00..127.56 rows=282608 width=17)
                       Output: t1.ls_pid_cusr1
                       Spwn on: All DataNodes
                       -> Seq Scan on channel.calc_empfyc_c1_result_tmp_t1 t1 (cost=0.00..1.62 rows=3947 width=17)
                           Output: t1.ls_pid_cusr1
                   SubPlan 1
                   -> Values Scan on "VALUES" (cost=0.00..0.01 rows=64 width=38)
                       Output: "VALUES".column1
```

Optimization

The test result shows that both result sets are too large. As a result, nestloop is time-consuming with more than one hour to return results. Therefore, the key to performance optimization is to eliminate nestloop, using more efficient hash join. From the perspective of semantic equivalence, the SQL statements can be written as follows:

```
selectls_pid_cusr1,COALESCE(max(round(ym/365)),0)from(
    (
        SELECT
        ls_pid_cusr1,(current_date-bthdate) as ym
        FROM calc_empfyc_c1_result_tmp_t1
        t1,p10_md_tmp_t2
        WHERE t1.ls_pid_cusr1 = t2.id and t1.ls_pid_cusr1 != t2.id15
    )
    union all
    (
        SELECT
        ls_pid_cusr1,(current_date-bthdate) as
        ym
        FROM calc_empfyc_c1_result_tmp_t1 t1,p10_md_tmp_t2 t2
        WHERE
        t1.ls_pid_cusr1 = id15
    ))GROUP BY ls_pid_cusr1;
```

The optimized SQL query consists of two equivalent join subqueries, and each subquery can be used for hash join in this scenario. The optimized execution plan is as follows.

id	operation	A-time	A-rows	E-rows	Peak Memory	Errors	A-width
1	-> Streaming (type: GATHER)	6737.281	0	192	292KB		
2	-> Insert on channel.calc_empfyc_cl_result_age_tmp	[4665.024,4990.666]	0	192	[1108KB, 1108KB]	1 MB	
3	-> HashAggregate	[4664.996,4990.641]	0	192	[12KB, 12KB]	1 MB	
4	-> Streaming (type: REDISTRIBUTE)	[4664.991,4990.637]	0	3392	[2090KB, 2090KB]	1 MB	
5	-> HashAggregate	[3416.939,4958.348]	0	3392	[14KB, 14KB]	1 MB	
6	-> Append	[3416.936,4958.340]	0	4011	[1KB, 1KB]	1 MB	
7	-> Hash Join (8,9)	[2011.226,3080.697]	0	3947	[6KB, 6KB]	1 MB	
8	-> Seq Scan on channel.p10_md_tmp_t2 t2	[803.782,1238.984]	443525717	443523360	[12KB, 12KB]	1 MB	
9	-> Hash	[4.357,328.979]	252608	252608	[482KB, 482KB]	1 MB	[58, 59]
10	-> Streaming (type: BROADCAST)	[2.345,326.320]	252608	252608	[2090KB, 2090KB]	1 MB	
11	-> Seq Scan on channel.calc_empfyc_cl_result_tmp_t1 t1	[0.011,0.030]	3947	3947	[11KB, 11KB]	1 MB	
12	-> Hash Join (13,14)	[1376.258,2066.110]	0	64	[5KB, 5KB]	1 MB	
13	-> Seq Scan on channel.p10_md_tmp_t2 t2	[777.552,1388.499]	443525717	443523360	[12KB, 12KB]	1 MB	
14	-> Hash	[2.812,4.217]	252608	252608	[482KB, 482KB]	1 MB	[58, 57]
15	-> Streaming (type: BROADCAST)	[1.276,1.868]	252608	252608	[2090KB, 2090KB]	1 MB	
16	-> Seq Scan on channel.calc_empfyc_cl_result_tmp_t1 t1	[0.010,0.033]	3947	3947	[11KB, 11KB]	1 MB	

Before the optimization, no result is returned for more than 1 hour. After the optimization, the result is returned within 7s.

6.13.8 Case: Modifying the GUC Parameter rewrite_rule

rewrite_rule contains multiple query rewriting rules: magicset, partialpush, uniquecheck, disablerep, intargetlist, and predpush. The following describes the application scenarios of some important rules.

Preparing the Case Environment

To demonstrate rule application scenarios, you need to prepare the following table creation statements:

```
-- Clean the environment.
DROP SCHEMA IF EXISTS rewrite_rule_guc_test CASCADE;
CREATE SCHEMA rewrite_rule_guc_test;
SET current_schema=rewrite_rule_guc_test;
-- Create a test table.
CREATE TABLE t(c1 INT, c2 INT, c3 INT, c4 INT);
CREATE TABLE t1(c1 INT, c2 INT, c3 INT, c4 INT);
CREATE TABLE t2(c1 INT, c2 INT, c3 INT, c4 INT);
```

partialpush: Partial Pushdown

Queries are pushed down to DN for distributed execution, greatly accelerating queries. If a query statement contains a factor that cannot be pushed down, the entire statement cannot be pushed down. As a result, a stream plan cannot be generated and executed on DN for the distributed execution, and the performance is poor.

The following is an example:

```
gaussdb=# set rewrite_rule='none';
SET
gaussdb=# explain (verbose on, costs off) select group_concat(tt.c1, tt.c2) from (select t1.c1,t2.c2 from t1,t2 where t1.c1=t2.c2) tt(c1,c2);
QUERY PLAN
-----
Aggregate
Output: group_concat(t1.c1, t2.c2 SEPARATOR ',')
-> Hash Join
Output: t1.c1, t2.c2
Hash Cond: (t1.c1 = t2.c2)
-> Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"
Output: t1.c1
Node/s: All datanodes
Remote query: SELECT c1 FROM ONLY public.t1 WHERE true
-> Hash
Output: t2.c2
-> Data Node Scan on t2 "_REMOTE_TABLE_QUERY_"
Output: t2.c2
```

```
Node/s: All datanodes
Remote query: SELECT c2 FROM ONLY public.t2 WHERE true
```

The group_concat() function cannot be pushed down. As a result, the remote query plan is executed:

1. Deliver the **select c1 from t1 where true** statement to DNs to read all data in the **t1** table.
2. Deliver the **select c2 from t2 where true** statement to DNs to read all data in the **t2** table.
3. Perform HASH JOIN on the CN.
4. Perform the group_concat calculation and return the final result.

This plan is slow because a large amount of data is transmitted over the network and then HASH JOIN is executed on the CN. As a result, cluster resources cannot be fully used.

partialpush is added to push the preceding 1, 2, and 3 operations down to DNs for distributed execution, greatly improving statement performance.

```
gaussdb=# set rewrite_rule='partialpush';
SET
gaussdb=# explain (verbose on, costs off) select two_sum(tt.c1, tt.c2) from (select t1.c1,t2.c2 from t1,t2
where t1.c1=t2.c2) tt(c1,c2);
QUERY PLAN
```

```
-----
Subquery Scan on tt
  Output: two_sum(tt.c1, tt.c2)
  -> Streaming (type: GATHER) -- The Gather plan is executed on DNs in a distributed manner:
    Output: t1.c1, t2.c2
    Node/s: All datanodes
    -> Nested Loop
      Output: t1.c1, t2.c2
      Join Filter: (t1.c1 = t2.c2)
      -> Seq Scan on public.t1
        Output: t1.c1, t1.c2, t1.c3
        Distribute Key: t1.c1
      -> Materialize
        Output: t2.c2
        -> Streaming(type: REDISTRIBUTE)
          Output: t2.c2
          Distribute Key: t2.c2
          Spawn on: All datanodes
          Consumer Nodes: All datanodes
        -> Seq Scan on public.t2
          Output: t2.c2
          Distribute Key: t2.c1
```

(21 rows)

intargetlist: Target Column Subquery Performance Improvement

The query performance can be greatly improved by converting the subquery in the target column to JOIN. The following is an example:

```
gaussdb=# set rewrite_rule='none';
SET
gaussdb=# explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2) from t1
where t1.c1<100 order by t1.c2;
QUERY PLAN
```

```
-----
Streaming (type: GATHER)
  Output: t1.c1, ((SubPlan 1)), t1.c2
  Merge Sort Key: t1.c2
  Node/s: All datanodes
```

```

-> Sort
  Output: t1.c1, ((SubPlan 1)), t1.c2
  Sort Key: t1.c2
  -> Seq Scan on public.t1
    Output: t1.c1, (SubPlan 1), t1.c2
    Distribute Key: t1.c1
    Filter: (t1.c1 < 100)
    SubPlan 1
      -> Aggregate
        Output: avg(t2.c2)
        -> Result
          Output: t2.c2
          Filter: (t2.c2 = t1.c2)
          -> Materialize
            Output: t2.c2
            -> Streaming(type: BROADCAST)
              Output: t2.c2
              Spawn on: All datanodes
              Consumer Nodes: All datanodes
            -> Seq Scan on public.t2
              Output: t2.c2
              Distribute Key: t2.c1
  (26 rows)

```

Because the subquery (**select avg(c2) from t2 where t2.c2=t1.c2**) in the target column cannot be pulled up, execution of the subquery is triggered each time a row of data of **t1** is scanned, and the query efficiency is low. If the **intargetlist** parameter is enabled, the subquery is converted to JOIN to improve the query performance.

```

gaussdb=# set rewrite_rule='intargetlist';
SET
gaussdb=# explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2) from t1
where t1.c1<100 order by t1.c2;
          QUERY PLAN
-----
Streaming (type: GATHER)
  Output: t1.c1, (avg(t2.c2)), t1.c2
  Merge Sort Key: t1.c2
  Node/s: All datanodes
  -> Sort
    Output: t1.c1, (avg(t2.c2)), t1.c2
    Sort Key: t1.c2
    -> Hash Right Join
      Output: t1.c1, (avg(t2.c2)), t1.c2
      Hash Cond: (t2.c2 = t1.c2)
      -> Streaming(type: BROADCAST)
        Output: (avg(t2.c2)), t2.c2
        Spawn on: All datanodes
        Consumer Nodes: All datanodes
      -> HashAggregate
        Output: avg(t2.c2), t2.c2
        Group By Key: t2.c2
        -> Streaming(type: REDISTRIBUTE)
          Output: t2.c2
          Distribute Key: t2.c2
          Spawn on: All datanodes
          Consumer Nodes: All datanodes
        -> Seq Scan on public.t2
          Output: t2.c2
          Distribute Key: t2.c1
    -> Hash
      Output: t1.c1, t1.c2
      -> Seq Scan on public.t1
        Output: t1.c1, t1.c2
        Distribute Key: t1.c1
        Filter: (t1.c1 < 100)
  (31 rows)

```

uniquecheck: Performance Improvement of Subqueries Without Aggregate Functions

Ensure that each condition has only one line of output. The subqueries with aggregate functions can be automatically pulled up. For subqueries without aggregate functions, the following is an example:

```
select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
```

Rewrite as follows:

```
select t1.c1 from t1 join (select t2.c1 from t2 where t2.c1 is not null group by t2.c1(unique check)) tt(c1) on tt.c1=t1.c1;
```

Note that unique check in the preceding SQL statement indicates that **t2.c1** needs to be checked. If the SQL statement is abnormal, the SQL statement cannot be directly executed. To ensure semantic equivalence, the subquery **tt** must ensure that each **group by t2.c1** has only one line of output. Enable the **uniquecheck** query rewriting parameter to ensure that the query can be pulled up and equivalent. If more than one row of data is output at run time, an error is reported.

```
gaussdb=# set rewrite_rule='uniquecheck';  
SET  
gaussdb=# explain verbose select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c1) ;  
QUERY PLAN
```

```
-----  
Streaming (type: GATHER)  
Output: t1.c1  
Node/s: All datanodes  
-> Nested Loop  
Output: t1.c1  
Join Filter: (t1.c1 = subquery."?column?")  
-> Seq Scan on public.t1  
Output: t1.c1, t1.c2, t1.c3  
Distribute Key: t1.c1  
-> Materialize  
Output: subquery."?column?", subquery.c1  
-> Subquery Scan on subquery  
Output: subquery."?column?", subquery.c1  
-> HashAggregate  
Output: t2.c1, t2.c1  
Group By Key: t2.c1  
Filter: (t2.c1 IS NOT NULL)  
Unique Check Required -- If more than one row of data is output during running, an  
error is reported.  
-> Index Only Scan using t2idx on public.t2  
Output: t2.c1  
Distribute Key: t2.c1
```

(21 rows)

Note: Because **group by t2.c1 unique check** occurs before the filter condition **tt.c1=t1.c1**, an error may be reported after the query that does not report an error is rewritten. An example is as follows:

There are tables **t1** and **t2**. The data in the tables is as follows:

```
gaussdb=# select * from t1 order by c2;  
c1 | c2 | c3  
----+----+----  
1 | 1 | 1  
2 | 2 | 2  
3 | 3 | 3  
4 | 4 | 4  
5 | 5 | 5  
6 | 6 | 6
```

```
7 | 7 | 7
8 | 8 | 8
9 | 9 | 9
10 | 10 | 10
(10 rows)
```

```
gaussdb=# select * from t2 order by c1;
```

```
c1 | c2 | c3
-----
1 | 1 | 1
2 | 2 | 2
3 | 3 | 3
4 | 4 | 4
5 | 5 | 5
6 | 6 | 6
7 | 7 | 7
8 | 8 | 8
9 | 9 | 9
10 | 10 | 10
11 | 11 | 11
11 | 11 | 11
12 | 12 | 12
12 | 12 | 12
13 | 13 | 13
13 | 13 | 13
14 | 14 | 14
14 | 14 | 14
15 | 15 | 15
15 | 15 | 15
16 | 16 | 16
16 | 16 | 16
17 | 17 | 17
17 | 17 | 17
18 | 18 | 18
18 | 18 | 18
19 | 19 | 19
19 | 19 | 19
20 | 20 | 20
20 | 20 | 20
(30 rows)
```

Disable and enable the **uniquecheck** parameter for comparison. After the parameter is enabled, an error is reported.

```
gaussdb=# select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
c1
```

```
----
6
7
3
1
2
4
5
8
9
10
(10 rows)
```

```
gaussdb=# set rewrite_rule='uniquecheck';
```

```
SET
```

```
gaussdb=# select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
ERROR: more than one row returned by a subquery used as an expression
```

predpush, predpushnormal, and predpushforce: Condition Pushdown to Subqueries

Generally, the optimizer performs optimization by query block, and different query blocks are independently optimized. If a predicate condition involving cross-query blocks exists, it is difficult to consider the location of a predicate application from a global perspective. The predpush may push down the predicate to the subquery block, so that performance can be improved in a scenario in which the data volume in the parent query block is relatively small and an index can be used in the subquery. There are three rewriting rules related to predpush:

- **predpushnormal**: attempts to push predicates down to subqueries. The STREAM operators, such as BROADCAST, are used to implement distributed plans.
- **predpushforce**: attempts to push down predicates to subqueries and uses the index of the parameterized path for scanning as much as possible.
- **predpush**: selects an optimal distributed plan from predpushnormal and predpushforce at a cost, but increases optimization time.

The following is an example of a plan for disabling and enabling the query rewriting rule:

```
gaussdb=# set enable_fast_query_shipping = off; -- Disable FQS optimization.
SET
gaussdb=# show rewrite_rule;
rewrite_rule
-----
magicset
(1 row)

gaussdb=# explain (costs off) select * from t1, (select sum(c2), c1 from t2 group by c1) st2 where st2.c1 =
t1.c1;
          QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Nested Loop
   Join Filter: (t1.c1 = t2.c1)
   -> HashAggregate
       Group By Key: t2.c1
       -> Seq Scan on t2
   -> Seq Scan on t1
(8 rows)

gaussdb=# set rewrite_rule='predpushnormal';
SET
gaussdb=# explain (costs off) select * from t1, (select sum(c2), c1 from t2 group by c1) st2 where st2.c1 =
t1.c1;
          QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Nested Loop
   -> Seq Scan on t1
   -> HashAggregate
       Group By Key: t2.c1
       -> Result
           Filter: (t1.c1 = t2.c1)
       -> Seq Scan on t2
(9 rows)

-- You can see that the filter criteria are pushed to the subquery for execution.
```

```
gaussdb=# set rewrite_rule='predpushforce';
SET

gaussdb=# explain (costs off) select * from t1, (select sum(c2), c1 from t2 group by c1) st2 where st2.c1 =
t1.c1;
          QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Nested Loop
   -> Seq Scan on t1
   -> HashAggregate
       Group By Key: t2.c1
       -> Index Scan using t2_c1_idx on t2
           Index Cond: (t1.c1 = c1)
(8 rows)

-- When used together with predpush hints, you can see that parameterized paths are used.

gaussdb=# set rewrite_rule = 'predpush';
SET
gaussdb=# explain (costs off) select * from t1, (select sum(c2), c1 from t2 group by c1) st2 where st2.c1 =
t1.c1;
          QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Nested Loop
   -> Seq Scan on t1
   -> HashAggregate
       Group By Key: t2.c1
       -> Result
           Filter: (t1.c1 = t2.c1)
           -> Seq Scan on t2
(9 rows)
```

Forbidding Pullup of Subquery Parameter `disablerep` for Replication Tables

When querying a replication table, the query actually takes effect on a DN only. Pulling up the subquery parameter **disablerep** may deteriorate the performance. The following is an example:

```
gaussdb=# create table t_rep(a int) distribute by replication;
CREATE TABLE
gaussdb=# create table t_dis(a int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# set rewrite_rule = '';
SET
gaussdb=# explain (costs off) select * from t_dis where a = any(select a from t_rep) or a > 100;
          QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Left Join
   Hash Cond: (t_dis.a = subquery.a)
   Filter: ((subquery.a IS NOT NULL) OR (t_dis.a > 100))
   -> Seq Scan on t_dis
   -> Hash
       -> Subquery Scan on subquery
           Filter: (Hash By subquery.a)
           -> HashAggregate
               Group By Key: t_rep.a
               -> Seq Scan on t_rep
(12 rows)
```

For a replication table, the data stored on all DN nodes is the same. Therefore, you do not need to scan the replication table on all nodes.

```
gaussdb=# set rewrite_rule = disablerep;
SET
gaussdb=# explain (costs off) select * from t_dis where a = any(select a from t_rep) or a > 100;
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on t_dis
    Filter: ((hashed SubPlan 1) OR (a > 100))
    SubPlan 1
        -> Seq Scan on t_rep
(6 rows)
```

6.13.9 Using DN Gather to Reduce Stream Nodes in the Plan

The DN Gather is used to remove the stream nodes from the distribution plan and send data to a node for calculation. This reduces the cost of data redistribution during the execution of the distribution plan and improves the single query performance and the overall throughput capability of the system. However, DN Gather is oriented to small-data-volume scenarios of TP. For small-data-volume queries, performance can be improved because the cost of data redistribution is reduced and the computing power of a single node is sufficient. Multi-node parallel computing is more advantageous for large-data-volume computing. You need to enable and disable DN Gather to determine which one is faster. (The default value of **dngather_min_rows** is **500**. The following uses the default value.) Some cases are provided as follows.

Preparing the Case Environment

To facilitate case demonstration, you need to prepare the following table creation statements:

```
-- Clean the environment.
DROP SCHEMA IF EXISTS dn_gather_test CASCADE;
CREATE SCHEMA dn_gather_test;
SET current_schema=dn_gather_test;
-- Create a test table.
CREATE TABLE t1(a INT, b INT, c INT, d INT);
CREATE TABLE t2(a INT, b INT, c INT, d INT);
CREATE TABLE t3(a INT, b INT, c INT, d INT);
CREATE TABLE t4(a INT, b INT, c INT, d INT);
```

Gather Join

To converge the join results to a single DN, the following conditions must be met:

- The number of data rows estimated by the optimizer before and after join is less than the threshold.
- The subnodes of Join are all stream nodes.

For example, the subnodes of Join are all stream nodes, and broadcast is disabled.

```
gaussdb=# set enable_broadcast=false;
SET
gaussdb=# set explain_perf_mode=pretty;
SET
gaussdb=# set enable_dngather=false;
SET
```

```

gaussdb=# explain select count(*) from t1, t2 where t1.b = t2.b;
id |          operation          | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Aggregate                |      1 |      8 | 31.46
2 | -> Streaming (type: GATHER) |      3 |      8 | 31.46
3 | -> Aggregate                |      3 |      8 | 31.34
4 | -> Hash Join (5,7)          |     30 |      0 | 31.30
5 | -> Streaming(type: REDISTRIBUTE) |     30 |      4 | 15.49
6 | -> Seq Scan on t1          |     30 |      4 | 14.14
7 | -> Hash                     |     29 |      4 | 15.49
8 | -> Streaming(type: REDISTRIBUTE) |     30 |      4 | 15.49
9 | -> Seq Scan on t2          |     30 |      4 | 14.14
(9 rows)

Predicate Information (identified by plan id)
-----
4 --Hash Join (5,7)
   Hash Cond: (t1.b = t2.b)
(2 rows)
gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select count(*) from t1, t2 where t1.b = t2.b;
id |          operation          | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) |      1 |      8 | 32.53
2 | -> Aggregate                |      1 |      8 | 32.47
3 | -> Hash Join (4,6)          |     30 |      0 | 32.38
4 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode1) |     30 |      4 | 15.69
5 | -> Seq Scan on t1          |     30 |      4 | 14.14
6 | -> Hash                     |     30 |      4 | 15.69
7 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode1) |     30 |      4 | 15.69
8 | -> Seq Scan on t2          |     30 |      4 | 14.14
(8 rows)

Predicate Information (identified by plan id)
-----
3 --Hash Join (4,6)
   Hash Cond: (t1.b = t2.b)
(2 rows)
gaussdb=# set enable_dngather=false;
SET
gaussdb=# explain select * from t1, t2, t3, t4 where t1.b = t2.b and t2.c = t3.c and t3.d = t4.d order by t1.a;
id |          operation          | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) |     30 |     144 | 66.46
2 | -> Sort                    |     30 |     144 | 65.05
3 | -> Hash Join (4,16)        |     30 |     144 | 64.86
4 | -> Streaming(type: REDISTRIBUTE) |     30 |     108 | 49.05
5 | -> Hash Join (6,13)        |     30 |     108 | 48.08
6 | -> Streaming(type: REDISTRIBUTE) |     30 |      72 | 32.27
7 | -> Hash Join (8,10)        |     30 |      72 | 31.30
8 | -> Streaming(type: REDISTRIBUTE) |     30 |      36 | 15.49
9 | -> Seq Scan on t1          |     30 |      36 | 14.14
10 | -> Hash                    |     29 |      36 | 15.49
11 | -> Streaming(type: REDISTRIBUTE) |     30 |      36 | 15.49
12 | -> Seq Scan on t2          |     30 |      36 | 14.14
13 | -> Hash                    |     29 |      36 | 15.49
14 | -> Streaming(type: REDISTRIBUTE) |     30 |      36 | 15.49
15 | -> Seq Scan on t3          |     30 |      36 | 14.14
16 | -> Hash                    |     29 |      36 | 15.49
17 | -> Streaming(type: REDISTRIBUTE) |     30 |      36 | 15.49
18 | -> Seq Scan on t4          |     30 |      36 | 14.14
(18 rows)

Predicate Information (identified by plan id)
-----
3 --Hash Join (4,16)
   Hash Cond: (t3.d = t4.d)
5 --Hash Join (6,13)

```

```

Hash Cond: (t2.c = t3.c)
7 --Hash Join (8,10)
Hash Cond: (t1.b = t2.b)
(6 rows)

gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select * from t1, t2, t3, t4 where t1.b = t2.b and t2.c = t3.c and t3.d = t4.d order by t1.a;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 144 | 68.47
2 | -> Sort | 30 | 144 | 66.36
3 | -> Hash Join (4,10) | 30 | 144 | 65.55
4 | -> Hash Join (5,7) | 30 | 72 | 32.38
5 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode3) | 30 | 36 | 15.69
6 | -> Seq Scan on t1 | 30 | 36 | 14.14
7 | -> Hash | 30 | 36 | 15.69
8 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode3) | 30 | 36 | 15.69
9 | -> Seq Scan on t2 | 30 | 36 | 14.14
10 | -> Hash | 30 | 72 | 32.38
11 | -> Hash Join (12,14) | 30 | 72 | 32.38
12 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode3) | 30 | 36 | 15.69
13 | -> Seq Scan on t3 | 30 | 36 | 14.14
14 | -> Hash | 30 | 36 | 15.69
15 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode3) | 30 | 36 | 15.69
16 | -> Seq Scan on t4 | 30 | 36 | 14.14
(16 rows)

Predicate Information (identified by plan id)
-----
3 --Hash Join (4,10)
Hash Cond: (t2.c = t3.c)
4 --Hash Join (5,7)
Hash Cond: (t1.b = t2.b)
11 --Hash Join (12,14)
Hash Cond: (t3.d = t4.d)
(6 rows)

gaussdb=# set enable_dngather=false;
SET
gaussdb=# explain select count(*) from t1, t2, t3, t4 where t1.b = t2.b and t2.c = t3.c and t3.d = t4.d group
by t1.b order by t1.b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 12 | 66.45
2 | -> GroupAggregate | 30 | 12 | 65.20
3 | -> Sort | 30 | 4 | 65.05
4 | -> Hash Join (5,17) | 30 | 4 | 64.86
5 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 49.05
6 | -> Hash Join (7,14) | 30 | 4 | 48.08
7 | -> Streaming(type: REDISTRIBUTE) | 30 | 8 | 32.27
8 | -> Hash Join (9,11) | 30 | 8 | 31.30
9 | -> Streaming(type: REDISTRIBUTE) | 30 | 8 | 15.49
10 | -> Seq Scan on t2 | 30 | 8 | 14.14
11 | -> Hash | 29 | 8 | 15.49
12 | -> Streaming(type: REDISTRIBUTE) | 30 | 8 | 15.49
13 | -> Seq Scan on t3 | 30 | 8 | 14.14
14 | -> Hash | 29 | 4 | 15.49
15 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 15.49
16 | -> Seq Scan on t4 | 30 | 4 | 14.14
17 | -> Hash | 29 | 4 | 15.49
18 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 15.49
19 | -> Seq Scan on t1 | 30 | 4 | 14.14
(19 rows)

Predicate Information (identified by plan id)
-----
4 --Hash Join (5,17)
Hash Cond: (t2.b = t1.b)
6 --Hash Join (7,14)

```

```

Hash Cond: (t3.d = t4.d)
8 --Hash Join (9,11)
Hash Cond: (t2.c = t3.c)
(6 rows)

gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select count(*) from t1, t2, t3, t4 where t1.b = t2.b and t2.c = t3.c and t3.d = t4.d group
by t1.b order by t1.b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 12 | 68.69
2 | -> GroupAggregate | 30 | 12 | 66.81
3 | -> Sort | 30 | 4 | 66.36
4 | -> Hash Join (5,11) | 30 | 4 | 65.55
5 | -> Hash Join (6,8) | 30 | 8 | 32.38
6 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode1) | 30 | 4 | 15.69
7 | -> Seq Scan on t1 | 30 | 4 | 14.14
8 | -> Hash | 30 | 8 | 15.69
9 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode1) | 30 | 8 | 15.69
10 | -> Seq Scan on t2 | 30 | 8 | 14.14
11 | -> Hash | 30 | 4 | 32.38
12 | -> Hash Join (13,15) | 30 | 4 | 32.38
13 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode1) | 30 | 8 | 15.69
14 | -> Seq Scan on t3 | 30 | 8 | 14.14
15 | -> Hash | 30 | 4 | 15.69
16 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode1) | 30 | 4 | 15.69
17 | -> Seq Scan on t4 | 30 | 4 | 14.14
(17 rows)

Predicate Information (identified by plan id)
-----
4 --Hash Join (5,11)
Hash Cond: (t2.c = t3.c)
5 --Hash Join (6,8)
Hash Cond: (t1.b = t2.b)
12 --Hash Join (13,15)
Hash Cond: (t3.d = t4.d)
(6 rows)

```

Gather Groupby/Agg

To converge the GroupBy/Agg results to a single DN, the following conditions must be met:

- The number of data rows estimated by the optimizer before and after GroupBy/Agg is less than the threshold.
- All agg subnodes are stream nodes.

```

gaussdb=# set explain_perf_mode=pretty;
SET
gaussdb=# set enable_dngather=false;
SET
gaussdb=# explain select count(*) from t1 group by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 12 | 15.87
2 | -> HashAggregate | 30 | 12 | 14.62
3 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 14.45
4 | -> Seq Scan on t1 | 30 | 4 | 14.14
(4 rows)

gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select count(*) from t1 group by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 12 | 16.85

```

```

2 | -> HashAggregate | 30 | 12 | 14.97
3 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode1) | 30 | 4 | 14.46
4 | -> Seq Scan on t1 | 30 | 4 | 14.14
(4 rows)

gaussdb=# set enable_dngather=false;
SET
gaussdb=# explain select b from t1 group by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 4 | 15.84
2 | -> HashAggregate | 30 | 4 | 14.59
3 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 14.45
4 | -> Seq Scan on t1 | 30 | 4 | 14.14
(4 rows)

gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select b from t1 group by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 4 | 16.74
2 | -> HashAggregate | 30 | 4 | 14.87
3 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode1) | 30 | 4 | 14.46
4 | -> Seq Scan on t1 | 30 | 4 | 14.14
(4 rows)

```

Gather Window Function

To converge window function results to a single DN, the following conditions must be met:

- The number of data rows estimated by the optimizer before and after the window function is less than the threshold.
- All subnodes of the window function are stream nodes.

```

gaussdb=# set explain_perf_mode=pretty;
SET
gaussdb=# set enable_dngather=false;
SET
gaussdb=# explain select count(*) over (partition by b) a from t1;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 29 | 4 | 16.71
2 | -> WindowAgg | 29 | 4 | 14.96
3 | -> Sort | 29 | 4 | 14.75
4 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 14.45
5 | -> Seq Scan on t1 | 30 | 4 | 14.14
(5 rows)

gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select count(*) over (partition by b) a from t1;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 4 | 19.07
2 | -> WindowAgg | 30 | 4 | 16.38
3 | -> Sort | 30 | 4 | 15.73
4 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode3) | 30 | 4 | 14.46
5 | -> Seq Scan on t1 | 30 | 4 | 14.14
(5 rows)

gaussdb=# set enable_dngather=false;
SET
gaussdb=# explain select sum(b) over (partition by b) a from t1 group by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 4 | 16.18

```

```

2 | -> WindowAgg          | 30 | 4 | 14.93
3 | -> Sort                | 30 | 4 | 14.78
4 | -> HashAggregate       | 30 | 4 | 14.59
5 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 14.45
6 | -> Seq Scan on t1     | 30 | 4 | 14.14
(6 rows)

gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select sum(b) over (partition by b) a from t1 group by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 30 | 4 | 18.00
2 | -> WindowAgg | 30 | 4 | 16.13
3 | -> Sort | 30 | 4 | 15.68
4 | -> HashAggregate | 30 | 4 | 14.87
5 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode3) | 30 | 4 | 14.46
6 | -> Seq Scan on t1 | 30 | 4 | 14.14
(6 rows)

```

Union/Union all

To converge union/union all results to a single DN, the following condition must be met:

- At least one subnode must meet the requirements in the three cases (Gather Join, Gather Groupby/Agg, and Gather window function).

For example, all the subnodes of Join are stream nodes, and broadcast is disabled.

```

gaussdb=# set explain_perf_mode=pretty;
SET
gaussdb=# set enable_broadcast=false;
SET
gaussdb=# set enable_dngather=false;
SET
gaussdb=# explain select t1.a, t2.b from t1, t2 where t1.b = t2.b union all select t3.a, t3.b from t3, t4 where t3.b = t4.b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 60 | 8 | 65.31
2 | -> Result | 60 | 8 | 62.81
3 | -> Append(4, 10) | 60 | 8 | 62.81
4 | -> Hash Join (5,7) | 30 | 8 | 31.30
5 | -> Streaming(type: REDISTRIBUTE) | 30 | 8 | 15.49
6 | -> Seq Scan on t1 | 30 | 8 | 14.14
7 | -> Hash | 29 | 4 | 15.49
8 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 15.49
9 | -> Seq Scan on t2 | 30 | 4 | 14.14
10 | -> Hash Join (11,13) | 30 | 8 | 31.30
11 | -> Streaming(type: REDISTRIBUTE) | 30 | 8 | 15.49
12 | -> Seq Scan on t3 | 30 | 8 | 14.14
13 | -> Hash | 29 | 4 | 15.49
14 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 15.49
15 | -> Seq Scan on t4 | 30 | 4 | 14.14
(15 rows)

Predicate Information (identified by plan id)
-----
4 --Hash Join (5,7)
Hash Cond: (t1.b = t2.b)
10 --Hash Join (11,13)
Hash Cond: (t3.b = t4.b)
(4 rows)

gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select t1.a, t2.b from t1, t2 where t1.b = t2.b union all select t3.a, t3.b from t3, t4 where

```

```
t3.b = t4.b;
```

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	60	8	69.11
2	-> Append(3, 9)	60	8	65.36
3	-> Hash Join (4,6)	30	8	32.38
4	-> Streaming(type: REDISTRIBUTE ng: node_group->datanode1)	30	8	15.69
5	-> Seq Scan on t1	30	8	14.14
6	-> Hash	30	4	15.69
7	-> Streaming(type: REDISTRIBUTE ng: node_group->datanode1)	30	4	15.69
8	-> Seq Scan on t2	30	4	14.14
9	-> Hash Join (10,12)	30	8	32.38
10	-> Streaming(type: REDISTRIBUTE ng: node_group->datanode1)	30	8	15.69
11	-> Seq Scan on t3	30	8	14.14
12	-> Hash	30	4	15.69
13	-> Streaming(type: REDISTRIBUTE ng: node_group->datanode1)	30	4	15.69
14	-> Seq Scan on t4	30	4	14.14

(14 rows)

Predicate Information (identified by plan id)

```
3 --Hash Join (4,6)
  Hash Cond: (t1.b = t2.b)
9 --Hash Join (10,12)
  Hash Cond: (t3.b = t4.b)
(4 rows)
```

```
gaussdb=# set enable_dngather=false;
SET
```

```
gaussdb=# explain select t1.a, t2.b from t1, t2 where t1.b = t2.b union select t3.a, t3.b from t3, t4 where
t3.b = t4.b order by a, b;
```

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	60	8	66.09
2	-> Sort	60	8	63.59
3	-> HashAggregate	60	8	63.11
4	-> Append(5, 11)	60	8	62.81
5	-> Hash Join (6,8)	30	8	31.30
6	-> Streaming(type: REDISTRIBUTE)	30	8	15.49
7	-> Seq Scan on t1	30	8	14.14
8	-> Hash	29	4	15.49
9	-> Streaming(type: REDISTRIBUTE)	30	4	15.49
10	-> Seq Scan on t2	30	4	14.14
11	-> Hash Join (12,14)	30	8	31.30
12	-> Streaming(type: REDISTRIBUTE)	30	8	15.49
13	-> Seq Scan on t3	30	8	14.14
14	-> Hash	29	4	15.49
15	-> Streaming(type: REDISTRIBUTE)	30	4	15.49
16	-> Seq Scan on t4	30	4	14.14

(16 rows)

Predicate Information (identified by plan id)

```
5 --Hash Join (6,8)
  Hash Cond: (t1.b = t2.b)
11 --Hash Join (12,14)
  Hash Cond: (t3.b = t4.b)
(4 rows)
```

```
gaussdb=# set enable_dngather=true;
SET
```

```
gaussdb=# explain select t1.a, t2.b from t1, t2 where t1.b = t2.b union select t3.a, t3.b from t3, t4 where
t3.b = t4.b order by a, b;
```

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	60	8	71.93
2	-> Sort	60	8	68.18
3	-> HashAggregate	60	8	66.26
4	-> Append(5, 11)	60	8	65.36

```

5 |      -> Hash Join (6,8) | 30 | 8 | 32.38
6 |      -> Streaming(type: REDISTRIBUTE ng: node_group->datanode2) | 30 | 8 | 15.69
7 |      -> Seq Scan on t1 | 30 | 8 | 14.14
8 |      -> Hash | 30 | 4 | 15.69
9 |      -> Streaming(type: REDISTRIBUTE ng: node_group->datanode2) | 30 | 4 | 15.69
10 |     -> Seq Scan on t2 | 30 | 4 | 14.14
11 |     -> Hash Join (12,14) | 30 | 8 | 32.38
12 |     -> Streaming(type: REDISTRIBUTE ng: node_group->datanode2) | 30 | 8 | 15.69
13 |     -> Seq Scan on t3 | 30 | 8 | 14.14
14 |     -> Hash | 30 | 4 | 15.69
15 |     -> Streaming(type: REDISTRIBUTE ng: node_group->datanode2) | 30 | 4 | 15.69
16 |     -> Seq Scan on t4 | 30 | 4 | 14.14
(16 rows)

```

Predicate Information (identified by plan id)

```

5 --Hash Join (6,8)
   Hash Cond: (t1.b = t2.b)
11 --Hash Join (12,14)
   Hash Cond: (t3.b = t4.b)
(4 rows)

```

gaussdb=# set enable_dngather=false;

SET

gaussdb=# explain select b, count(*) from t1 group by b union all select b, count(*) from t2 group by b order by b;

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	60	12	32.43
2	-> Sort	60	12	29.93
3	-> Result	60	12	29.45
4	-> Append(5, 8)	60	12	29.45
5	-> HashAggregate	30	12	14.62
6	-> Streaming(type: REDISTRIBUTE)	30	4	14.45
7	-> Seq Scan on t1	30	4	14.14
8	-> HashAggregate	30	12	14.62
9	-> Streaming(type: REDISTRIBUTE)	30	4	14.45
10	-> Seq Scan on t2	30	4	14.14

(10 rows)

gaussdb=# set enable_dngather=true;

SET

gaussdb=# explain select b, count(*) from t1 group by b union all select b, count(*) from t2 group by b order by b;

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	60	12	36.22
2	-> Sort	60	12	32.47
3	-> Append(4, 7)	60	12	30.55
4	-> HashAggregate	30	12	14.97
5	-> Streaming(type: REDISTRIBUTE ng: node_group->datanode2)	30	4	14.46
6	-> Seq Scan on t1	30	4	14.14
7	-> HashAggregate	30	12	14.97
8	-> Streaming(type: REDISTRIBUTE ng: node_group->datanode2)	30	4	14.46
9	-> Seq Scan on t2	30	4	14.14

(9 rows)

gaussdb=# set enable_dngather=false;

SET

gaussdb=# explain select b, count(*) from t1 group by b union all select count(distinct a) a , count(distinct b)b from t2 order by b;

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	33	12	20000000045.02
2	-> Sort	33	12	20000000043.65
3	-> Append(4, 8)	33	12	20000000043.43
4	-> Subquery Scan on ""SELECT* 1"	30	12	14.72
5	-> HashAggregate	30	12	14.62
6	-> Streaming(type: REDISTRIBUTE)	30	4	14.45

```

7 |      -> Seq Scan on t1 | 30 | 4 | 14.14
8 |    -> Subquery Scan on "**SELECT* 2" | 1 | 16 | 20000000028.73
9 |      -> Nested Loop (10,14) | 3 | 16 | 20000000028.70
10 |        -> Aggregate | 3 | 12 | 10000000014.18
11 |          -> Streaming(type: BROADCAST) | 9 | 12 | 10000000014.18
12 |            -> Aggregate | 3 | 12 | 14.19
13 |              -> Seq Scan on t2 | 30 | 4 | 14.14
14 |                -> Materialize | 3 | 8 | 10000000014.49
15 |                  -> Aggregate | 3 | 12 | 10000000014.48
16 |                    -> Streaming(type: BROADCAST) | 9 | 12 | 10000000014.48
17 |                      -> Aggregate | 3 | 12 | 14.48
18 |                        -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 14.45
19 |                          -> Seq Scan on t2 | 30 | 4 | 14.14
(19 rows)

Predicate Information (identified by plan id)
-----
 8 --Subquery Scan on "**SELECT* 2"
      Filter: (Hash By "**SELECT* 2".a)
(2 rows)

gaussdb=# set enable_dngather=true;
SET
gaussdb=# explain select b, count(*) from t1 group by b union all select count(distinct a) a , count(distinct
b)b from t2 order by b;
id | operation | E-rows | E-width | E-costs
-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 33 | 11 | 20000000046.96
2 | -> Sort | 33 | 11 | 20000000044.90
3 | -> Append(4, 8) | 33 | 11 | 20000000043.99
4 | -> Subquery Scan on "**SELECT* 1" | 30 | 12 | 15.27
5 | -> HashAggregate | 30 | 12 | 14.97
6 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode2) | 30 | 4 | 14.46
7 | -> Seq Scan on t1 | 30 | 4 | 14.14
8 | -> Streaming(type: REDISTRIBUTE ng: node_group->datanode2) | 3 | 16 |
20000000028.83
9 | -> Nested Loop (10,14) | 3 | 16 | 20000000028.70
10 | -> Aggregate | 3 | 12 | 10000000014.18
11 | -> Streaming(type: BROADCAST) | 9 | 12 | 10000000014.18
12 | -> Aggregate | 3 | 12 | 14.19
13 | -> Seq Scan on t2 | 30 | 4 | 14.14
14 | -> Materialize | 3 | 8 | 10000000014.50
15 | -> Aggregate | 3 | 12 | 10000000014.48
16 | -> Streaming(type: BROADCAST) | 9 | 12 | 10000000014.48
17 | -> Aggregate | 3 | 12 | 14.48
18 | -> Streaming(type: REDISTRIBUTE) | 30 | 4 | 14.45
19 | -> Seq Scan on t2 | 30 | 4 | 14.14
(19 rows)

```

7 SQL Reference

7.1 SQL

What Is SQL?

SQL is a standard computer language used to control the access to databases and manage data in databases.

SQL provides different statements to enable you to:

- Query data.
- Insert, update, and delete rows.
- Create, replace, modify, and delete objects.
- Control the access to a database and its objects.
- Maintain the consistency and integrity of a database.

SQL consists of commands and functions that are used to manage databases and database objects. This language can also forcibly implement the rules for data types, expressions, and texts. Therefore, [SQL Reference](#) describes data types, expressions, functions, and operators in addition to SQL syntax.

Development of SQL Standards

Released SQL standards are as follows:

- 1986: ANSI X3.135-1986, ISO/IEC 9075:1986, SQL-86
- 1989: ANSI X3.135-1989, ISO/IEC 9075:1989, SQL-89
- 1992: ANSI X3.135-1992, ISO/IEC 9075:1992, SQL-92
- 1999: ISO/IEC 9075:1999, SQL:1999
- 2003: ISO/IEC 9075:2003, SQL:2003
- 2011: ISO/IEC 9075:2011, SQL:2011
- 2016: ISO/IEC 9075:2016, SQL:2016
- 2019: ISO/IEC 9075:2019, SQL:2019

SQL Standards Supported by GaussDB

By default, GaussDB supports most features of SQL:2016.

7.2 Keywords

The SQL contains reserved keywords and non-reserved keywords. Standards require that reserved keywords not be used as other identifiers. Non-reserved keywords have special meanings only in a specific environment and can be used as identifiers in other environments.

NOTICE

1. Currently, the non-reserved keywords have the following restrictions when being used as the identifier of a database object:
 1. It cannot be directly used as a column alias. That is, usage similar to `SELECT 1 ABORT` may cause errors.
 2. Keywords `ENTITYESCAPING`, `NOENTITYESCAPING`, and `WELLFORMED` cannot be used as identifiers of table names, column names, table aliases, column aliases, and function names if they are not enclosed in double quotation marks.
 3. The `RAW` keyword without double quotation marks cannot be used as the identifier of a table name or function name.
 4. The `SET` keyword without double quotation marks cannot be used as an identifier of a table alias. That is, usage similar to `SELECT * FROM T1 SET` may cause errors.
 5. Keywords such as `BEGIN`, `BY`, `CLOSE`, `CURSOR`, `DECLARE`, `DELETE`, `EXECUTE`, `FUNCTION`, `IF`, `IMMEDIATE`, `INSERT`, `LOOP`, `MOVE`, `OF`, `REF`, `RELEASE`, `RETURN`, `SAVEPOINT`, `STRICT`, `TYPE`, and `UPDATE` without double quotation marks cannot be used as variable names.
 6. When the `SYS_REFCURSOR` keyword is used as the identifier of a database object, if double quotation marks are not attached, a database object named **REFCURSOR** is created. If double quotation marks are attached, a database object named **SYS_REFCURSOR** is created.
 2. Similar to the non-reserved keywords, the non-reserved (cannot be a function or type) keywords cannot be directly used as column aliases, either.
 3. The reserved keyword `CURRENT_TIMESTAMP` without double quotation marks (""") cannot be used as a function name.
-

Identifier Naming Conventions

Identifier naming must comply with the following rules:

- An identifier name can only contain letters, digits, underscores (`_`), and dollar signs (`$`).
- An identifier name must start with a letter or an underscore (`_`).

 NOTE

- The naming rules are recommended but not required.
- In special cases, double quotation marks (") can be used to avoid special character errors.

SQL Keywords

Table 7-1 SQL keywords

Keyword	GaussDB	SQL:1999	SQL-92
ABORT	Non-reserved	N/A	N/A
ABS	N/A	Non-reserved	N/A
ABSOLUTE	Non-reserved	Reserved	Reserved
ACCESS	Non-reserved	N/A	N/A
ACCOUNT	Non-reserved	N/A	N/A
ACTION	Non-reserved	Reserved	Reserved
ADA	N/A	Non-reserved	Non-reserved
ADD	Non-reserved	Reserved	Reserved
ADMIN	Non-reserved	Reserved	N/A
AFTER	Non-reserved	Reserved	N/A
AGGREGATE	Non-reserved	Reserved	N/A
ALGORITHM	Non-reserved	N/A	N/A
ALIAS	N/A	Reserved	N/A
ALL	Reserved	Reserved	Reserved
ALLOCATE	N/A	Reserved	Reserved
ALSO	Non-reserved	N/A	N/A
ALTER	Non-reserved	Reserved	Reserved
ALWAYS	Non-reserved	N/A	N/A
ANALYSE	Reserved	N/A	N/A
ANALYZE	Reserved	N/A	N/A
AND	Reserved	Reserved	Reserved
ANY	Reserved	Reserved	Reserved
APP	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
APPEND	Non-reserved	N/A	N/A
ARCHIVE	Non-reserved	N/A	N/A
ARE	N/A	Reserved	Reserved
ARRAY	Reserved	Reserved	N/A
AS	Reserved	Reserved	Reserved
ASC	Reserved	Reserved	Reserved
ASENSITIVE	N/A	Non-reserved	N/A
ASSERTION	Non-reserved	Reserved	Reserved
ASSIGNMENT	Non-reserved	Non-reserved	N/A
ASYMMETRIC	Reserved	Non-reserved	N/A
AT	Non-reserved	Reserved	Reserved
ATOMIC	N/A	Non-reserved	N/A
ATTRIBUTE	Non-reserved	N/A	N/A
AUDIT	Non-reserved	N/A	N/A
AUTHID	Reserved	N/A	N/A
AUTHORIZATION	Reserved (functions and types allowed)	Reserved	Reserved
AUTO_INCREMENT	Non-reserved	N/A	N/A
AUTOEXTEND	Non-reserved	N/A	N/A
AUTOMAPPED	Non-reserved	N/A	N/A
AVG	N/A	Non-reserved	Reserved
BACKWARD	Non-reserved	N/A	N/A
BAD_PATH	Non-reserved	N/A	N/A
BARRIER	Non-reserved	N/A	N/A
BEFORE	Non-reserved	Reserved	N/A
BEGIN	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
BEGIN_NON_ANOYBLOCK	Non-reserved	N/A	N/A
BETWEEN	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
BIGINT	Non-reserved (cannot be functions or types)	N/A	N/A
BINARY	Reserved (functions and types allowed)	Reserved	N/A
BINARY_DOUBLE	Non-reserved (cannot be functions or types)	N/A	N/A
BINARY_INTEGER	Non-reserved (cannot be functions or types)	N/A	N/A
BIT	Non-reserved (cannot be functions or types)	Reserved	Reserved
BIT_LENGTH	N/A	Non-reserved	Reserved
BITVAR	N/A	Non-reserved	N/A
BLANKS	Non-reserved	N/A	N/A
BLOB	Non-reserved	Reserved	N/A
BODY	Non-reserved	N/A	N/A
BOOLEAN	Non-reserved (cannot be functions or types)	Reserved	N/A
BOTH	Reserved	Reserved	Reserved
BREADTH	N/A	Reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
BUCKETCNT	Non-reserved (cannot be functions or types)	N/A	N/A
BUCKETS	Reserved	N/A	N/A
BY	Non-reserved	Reserved	Reserved
BYTEAWITHOUTORDER	Non-reserved (cannot be functions or types)	N/A	N/A
BYTEAWITHOUTORDER-WITHEQUAL	Non-reserved (cannot be functions or types)	N/A	N/A
C	N/A	Non-reserved	Non-reserved
CACHE	Non-reserved	N/A	N/A
CALL	Non-reserved	Reserved	N/A
CALLED	Non-reserved	Non-reserved	N/A
CANCELABLE	Non-reserved	N/A	N/A
CARDINALITY	N/A	Non-reserved	N/A
CASCADE	Non-reserved	Reserved	Reserved
CASCADEED	Non-reserved	Reserved	Reserved
CASE	Reserved	Reserved	Reserved
CAST	Reserved	Reserved	Reserved
CATALOG	Non-reserved	Reserved	Reserved
CATALOG_NAME	N/A	Non-reserved	Non-reserved
CHAIN	Non-reserved	Non-reserved	N/A
CHANGE	Non-reserved	N/A	N/A
CHAR	Non-reserved (cannot be functions or types)	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
CHAR_LENGTH	N/A	Non-reserved	Reserved
CHARACTER	Non-reserved (cannot be functions or types)	Reserved	Reserved
CHARACTER_LENGTH	N/A	Non-reserved	Reserved
CHARACTER_SET_CATALOG	N/A	Non-reserved	Non-reserved
CHARACTER_SET_NAME	N/A	Non-reserved	Non-reserved
CHARACTER_SET_SCHEMA	N/A	Non-reserved	Non-reserved
CHARACTERISTICS	Non-reserved	N/A	N/A
CHARACTERSET	Non-reserved	N/A	N/A
CHARSET	Non-reserved	N/A	N/A
CHECK	Reserved	Reserved	Reserved
CHECKED	N/A	Non-reserved	N/A
CHECKPOINT	Non-reserved	N/A	N/A
CLASS	Non-reserved	Reserved	N/A
CLASS_ORIGIN	N/A	Non-reserved	Non-reserved
CLEAN	Non-reserved	N/A	N/A
CLIENT	Non-reserved	N/A	N/A
CLIENT_MASTER_KEY	Non-reserved	N/A	N/A
CLIENT_MASTER_KEYS	Non-reserved	N/A	N/A
CLOB	Non-reserved	Reserved	N/A
CLOSE	Non-reserved	Reserved	Reserved
CLUSTER	Non-reserved	N/A	N/A
COALESCE	Non-reserved (cannot be functions or types)	Non-reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
COBOL	N/A	Non-reserved	Non-reserved
COLLATE	Reserved	Reserved	Reserved
COLLATION	Reserved (functions and types allowed)	Reserved	Reserved
COLLATION_CATALOG	N/A	Non-reserved	Non-reserved
COLLATION_NAME	N/A	Non-reserved	Non-reserved
COLLATION_SCHEMA	N/A	Non-reserved	Non-reserved
COLUMN	Reserved	Reserved	Reserved
COLUMN_ENCRYPTION_KEY	Non-reserved	N/A	N/A
COLUMN_ENCRYPTION_KEYS	Non-reserved	N/A	N/A
COLUMN_NAME	N/A	Non-reserved	Non-reserved
COLUMNS	Non-reserved	N/A	N/A
COMMAND_FUNCTION	N/A	Non-reserved	Non-reserved
COMMAND_FUNCTION_CODE	N/A	Non-reserved	N/A
COMMENT	Non-reserved	N/A	N/A
COMMENTS	Non-reserved	N/A	N/A
COMMIT	Non-reserved	Reserved	Reserved
COMMITTED	Non-reserved	Non-reserved	Non-reserved
COMPACT	Reserved (functions and types allowed)	N/A	N/A
COMPATIBLE_ILLEGAL_CHARS	Non-reserved	N/A	N/A
COMPILE	Non-reserved	N/A	N/A
COMPLETE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
COMPLETION	Non-reserved	Reserved	N/A
COMPRESS	Non-reserved	N/A	N/A
CONCURRENTLY	Reserved (functions and types allowed)	N/A	N/A
CONDITION	Non-reserved	N/A	N/A
CONDITION_NUMBER	N/A	Non-reserved	Non-reserved
CONFIGURATION	Non-reserved	N/A	N/A
CONNECT	Non-reserved	Reserved	Reserved
CONNECTION	Non-reserved	Reserved	Reserved
CONNECTION_NAME	N/A	Non-reserved	Non-reserved
CONSTANT	Non-reserved	N/A	N/A
CONSTRAINT	Reserved	Reserved	Reserved
CONSTRAINT_CATALOG	N/A	Non-reserved	Non-reserved
CONSTRAINT_NAME	N/A	Non-reserved	Non-reserved
CONSTRAINT_SCHEMA	N/A	Non-reserved	Non-reserved
CONSTRAINTS	Non-reserved	Reserved	Reserved
CONSTRUCTOR	N/A	Reserved	N/A
CONTAINING	Non-reserved	N/A	N/A
CONTAINS	N/A	Non-reserved	N/A
CONTENT	Non-reserved	N/A	N/A
CONTINUE	Non-reserved	Reserved	Reserved
CONVERSION	Non-reserved	N/A	N/A
CONVERT	Non-reserved	Non-reserved	Reserved
COORDINATOR	Non-reserved	N/A	N/A
COORDINATORS	Non-reserved	N/A	N/A
COPY	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
CORRESPONDING	N/A	Reserved	Reserved
COST	Non-reserved	N/A	N/A
COUNT	N/A	Non-reserved	Reserved
CREATE	Reserved	Reserved	Reserved
CROSS	Reserved (functions and types allowed)	Reserved	Reserved
CSN	Reserved (functions and types allowed)	N/A	N/A
CSV	Non-reserved	N/A	N/A
CUBE	Non-reserved	Reserved	N/A
CURRENT	Non-reserved	Reserved	Reserved
CURRENT_CATALOG	Reserved	N/A	N/A
CURRENT_DATE	Reserved	Reserved	Reserved
CURRENT_PATH	N/A	Reserved	N/A
CURRENT_ROLE	Reserved	Reserved	N/A
CURRENT_SCHEMA	Reserved (functions and types allowed)	N/A	N/A
CURRENT_TIME	Reserved	Reserved	Reserved
CURRENT_TIMESTAMP	Reserved	Reserved	Reserved
CURRENT_USER	Reserved	Reserved	Reserved
CURSOR	Non-reserved	Reserved	Reserved
CURSOR_NAME	N/A	Non-reserved	Non-reserved
CYCLE	Non-reserved	Reserved	N/A
DATA	Non-reserved	Reserved	Non-reserved
DATABASE	Non-reserved	N/A	N/A
DATAFILE	Non-reserved	N/A	N/A
DATANODE	Non-reserved	N/A	N/A
DATANODES	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
DATATYPE_CL	Non-reserved	N/A	N/A
DATE	Non-reserved (cannot be functions or types)	Reserved	Reserved
DATE_FORMAT	Non-reserved	N/A	N/A
DATETIME_INTERVAL_CODE	N/A	Non-reserved	Non-reserved
DATETIME_INTERVAL_PRECISION	N/A	Non-reserved	Non-reserved
DAY	Non-reserved	Reserved	Reserved
DB4AISHOT	Non-reserved	N/A	N/A
DBCOMPATIBILITY	Non-reserved	N/A	N/A
DBTIMEZONE	Reserved	N/A	N/A
DEALLOCATE	Non-reserved	Reserved	Reserved
DEC	Non-reserved (cannot be functions or types)	Reserved	Reserved
DECIMAL	Non-reserved (cannot be functions or types)	Reserved	Reserved
DECLARE	Non-reserved	Reserved	Reserved
DECODE	Non-reserved (cannot be functions or types)	N/A	N/A
DEFAULT	Reserved	Reserved	Reserved
DEFAULTS	Non-reserved	N/A	N/A
DEFERRABLE	Reserved	Reserved	Reserved
DEFERRED	Non-reserved	Reserved	Reserved
DEFINED	N/A	Non-reserved	N/A
DEFINER	Non-reserved	Non-reserved	N/A
DELETE	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
DELIMITER	Non-reserved	N/A	N/A
DELIMITERS	Non-reserved	N/A	N/A
DELTA	Non-reserved	N/A	N/A
DELTAMERGE	Reserved (functions and types allowed)	N/A	N/A
DEPTH	N/A	Reserved	N/A
DEREF	N/A	Reserved	N/A
DESC	Reserved	Reserved	Reserved
DESCRIBE	N/A	Reserved	Reserved
DESCRIPTOR	N/A	Reserved	Reserved
DESTROY	N/A	Reserved	N/A
DESTRUCTOR	N/A	Reserved	N/A
DETERMINISTIC	Non-reserved	Reserved	N/A
DIAGNOSTICS	N/A	Reserved	Reserved
DICTIONARY	Non-reserved	Reserved	N/A
DIRECT	Non-reserved	N/A	N/A
DIRECTORY	Non-reserved	N/A	N/A
DISABLE	Non-reserved	N/A	N/A
DISCARD	Non-reserved	N/A	N/A
DISCARD_PATH	Non-reserved	N/A	N/A
DISCONNECT	Non-reserved	Reserved	Reserved
DISPATCH	N/A	Non-reserved	N/A
DISTINCT	Reserved	Reserved	Reserved
DISTRIBUTE	Non-reserved	N/A	N/A
DISTRIBUTION	Non-reserved	N/A	N/A
DO	Reserved	N/A	N/A
DOCUMENT	Non-reserved	N/A	N/A
DOMAIN	Non-reserved	Reserved	Reserved
DOUBLE	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
DROP	Non-reserved	Reserved	Reserved
DUMPFIL	Non-reserved	N/A	N/A
DUPLICATE	Non-reserved	N/A	N/A
DYNAMIC	N/A	Reserved	N/A
DYNAMIC_FUNCTION	N/A	Non-reserved	Non-reserved
DYNAMIC_FUNCTION_CODE	N/A	Non-reserved	N/A
EACH	Non-reserved	Reserved	N/A
ELASTIC	Non-reserved	N/A	N/A
ELSE	Reserved	Reserved	Reserved
ENABLE	Non-reserved	N/A	N/A
ENCLOSED	Non-reserved	N/A	N/A
ENCODING	Non-reserved	N/A	N/A
ENCRYPTED	Non-reserved	N/A	N/A
ENCRYPTED_VALUE	Non-reserved	N/A	N/A
ENCRYPTION	Non-reserved	N/A	N/A
ENCRYPTION_TYPE	Non-reserved	N/A	N/A
END	Reserved	Reserved	Reserved
END-EXEC	N/A	Reserved	Reserved
ENDS	Non-reserved	N/A	N/A
ENFORCED	Non-reserved	N/A	N/A
ENTITYESCAPING	Non-reserved	N/A	N/A
ENUM	Non-reserved	N/A	N/A
EOL	Non-reserved	N/A	N/A
EQUALS	N/A	Reserved	N/A
ERROR	Non-reserved	N/A	N/A
ERRORS	Non-reserved	N/A	N/A
ESCAPE	Non-reserved	Reserved	Reserved
ESCAPED	Non-reserved	N/A	N/A
ESCAPING	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
EVALNAME	Non-reserved	N/A	N/A
EVENT	Non-reserved	N/A	N/A
EVENTS	Non-reserved	N/A	N/A
EVERY	Non-reserved	Reserved	N/A
EXCEPT	Reserved	Reserved	Reserved
EXCEPTION	N/A	Reserved	Reserved
EXCHANGE	Non-reserved	N/A	N/A
EXCLUDE	Non-reserved	N/A	N/A
EXCLUDED	Reserved	N/A	N/A
EXCLUDING	Non-reserved	N/A	N/A
EXCLUSIVE	Non-reserved	N/A	N/A
EXEC	N/A	Reserved	Reserved
EXECUTE	Non-reserved	Reserved	Reserved
EXISTING	N/A	Non-reserved	N/A
EXISTS	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
EXPDP	Non-reserved	N/A	N/A
EXPIRED_P	N/A	N/A	N/A
EXPLAIN	Non-reserved	N/A	N/A
EXTENSION	Non-reserved	N/A	N/A
EXTERNAL	Non-reserved	Reserved	Reserved
EXTRACT	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
FALSE	Reserved	Reserved	Reserved
FAMILY	Non-reserved	N/A	N/A
FAST	Non-reserved	N/A	N/A
FEATURES	Non-reserved	N/A	N/A
FENCED	Reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
FETCH	Reserved	Reserved	Reserved
FIELDS	Non-reserved	N/A	N/A
FILEHEADER	Non-reserved	N/A	N/A
FILL_MISSING_FIELDS	Non-reserved	N/A	N/A
FILLER	Non-reserved	N/A	N/A
FILTER	Non-reserved	N/A	Reserved
FINAL	N/A	Non-reserved	N/A
FIRST	Non-reserved	Reserved	Reserved
FIXED	Non-reserved	N/A	Reserved
FLOAT	Non-reserved (cannot be functions or types)	Reserved	Reserved
FOLLOWING	Non-reserved	N/A	N/A
FOR	Reserved	Reserved	Reserved
FORCE	Non-reserved	N/A	N/A
FOREIGN	Reserved	Reserved	Reserved
FORMATTER	Non-reserved	N/A	N/A
FORTRAN	N/A	Non-reserved	Non-reserved
FORWARD	Non-reserved	N/A	N/A
FOUND	N/A	Reserved	Reserved
FREE	N/A	Reserved	N/A
FREEZE	Reserved (functions and types allowed)	N/A	N/A
FROM	Reserved	Reserved	Reserved
FULL	Reserved (functions and types allowed)	Reserved	Reserved
FUNCTION	Non-reserved	Reserved	N/A
FUNCTIONS	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
G	N/A	Non-reserved	N/A
GENERAL	N/A	Reserved	N/A
GENERATED	Non-reserved	Non-reserved	N/A
GET	N/A	Reserved	Reserved
GLOBAL	Non-reserved	Reserved	Reserved
GO	N/A	Reserved	Reserved
GOTO	N/A	Reserved	Reserved
GRANT	Reserved	Reserved	Reserved
GRANTED	Non-reserved	Non-reserved	N/A
GREATEST	Non-reserved (cannot be functions or types)	N/A	N/A
GROUP	Reserved	Reserved	Reserved
GROUPING	Non-reserved (cannot be functions or types)	Reserved	N/A
GROUPPARENT	Reserved	N/A	N/A
HANDLER	Non-reserved	N/A	N/A
HAVING	Reserved	Reserved	Reserved
HDFSDIRECTORY	Reserved (functions and types allowed)	N/A	N/A
HEADER	Non-reserved	N/A	N/A
HIERARCHY	N/A	Non-reserved	N/A
HOLD	Non-reserved	Non-reserved	N/A
HOST	N/A	Reserved	N/A
HOUR	Non-reserved	Reserved	Reserved
IDENTIFIED	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
IDENTITY	Non-reserved	Reserved	Reserved
IF	Non-reserved	N/A	N/A
IGNORE	Non-reserved	Reserved	N/A
IGNORE_EXTRA_DATA	Non-reserved	N/A	N/A
ILIKE	Reserved (functions and types allowed)	N/A	N/A
IMMEDIATE	Non-reserved	Reserved	Reserved
IMMUTABLE	Non-reserved	N/A	N/A
IMPDP	Non-reserved	N/A	N/A
IMPLEMENTATION	N/A	Non- reserved	N/A
IMPLICIT	Non-reserved	N/A	N/A
IN	Reserved	Reserved	Reserved
INCLUDE	Non-reserved	N/A	N/A
INCLUDING	Non-reserved	N/A	N/A
INCREMENT	Non-reserved	N/A	N/A
INCREMENTAL	Non-reserved	N/A	N/A
INDEX	Non-reserved	N/A	N/A
INDEXES	Non-reserved	N/A	N/A
INDICATOR	N/A	Reserved	Reserved
INFILE	Non-reserved	N/A	N/A
INFIX	N/A	Non- reserved	N/A
INHERIT	Non-reserved	N/A	N/A
INHERITS	Non-reserved	N/A	N/A
INITIAL	Non-reserved	N/A	N/A
INITIALIZE	N/A	Reserved	N/A
INITIALLY	Reserved	Reserved	Reserved
INITRANS	Non-reserved	N/A	N/A
INLINE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
INNER	Reserved (functions and types allowed)	Reserved	Reserved
INOUT	Non-reserved (cannot be functions or types)	Reserved	N/A
INPUT	Non-reserved	Reserved	Reserved
INSENSITIVE	Non-reserved	Non-reserved	Reserved
INSERT	Non-reserved	Reserved	Reserved
INSTANCE	N/A	Non-reserved	N/A
INSTANTIABLE	N/A	Non-reserved	N/A
INSTEAD	Non-reserved	N/A	N/A
INT	Non-reserved (cannot be functions or types)	Reserved	Reserved
INTEGER	Non-reserved (cannot be functions or types)	Reserved	Reserved
INTERNAL	Non-reserved	N/A	N/A
INTERSECT	Reserved	Reserved	Reserved
INTERVAL	Non-reserved (cannot be functions or types)	Reserved	Reserved
INTO	Reserved	Reserved	Reserved
INVOKER	Non-reserved	Non-reserved	N/A
IP	Non-reserved	N/A	N/A
IS	Reserved	Reserved	Reserved
ISNULL	Non-reserved	N/A	N/A
ISOLATION	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
ITERATE	N/A	Reserved	N/A
JOIN	Reserved (functions and types allowed)	Reserved	Reserved
K	N/A	Non- reserved	N/A
KEY	Non-reserved	Reserved	Reserved
KEY_MEMBER	N/A	Non- reserved	N/A
KEY_PATH	Non-reserved	N/A	N/A
KEY_STORE	Non-reserved	N/A	N/A
KEY_TYPE	N/A	Non- reserved	N/A
KILL	Non-reserved	N/A	N/A
LABEL	Non-reserved	N/A	N/A
LANGUAGE	Non-reserved	Reserved	Reserved
LARGE	Non-reserved	Reserved	N/A
LAST	Non-reserved	Reserved	Reserved
LATERAL	N/A	Reserved	N/A
LC_COLLATE	Non-reserved	N/A	N/A
LC_CTYPE	Non-reserved	N/A	N/A
LEADING	Reserved	Reserved	Reserved
LEAKPROOF	Non-reserved	N/A	N/A
LEAST	Non-reserved (cannot be functions or types)	N/A	N/A
LEFT	Reserved (functions and types allowed)	Reserved	Reserved
LENGTH	N/A	Non- reserved	Non-reserved
LESS	Reserved	Reserved	N/A
LEVEL	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
LIKE	Reserved (functions and types allowed)	Reserved	Reserved
LIMIT	Reserved	Reserved	N/A
LINES	Non-reserved	N/A	N/A
LINK	Non-reserved	N/A	N/A
LIST	Non-reserved	N/A	N/A
LISTEN	Non-reserved	N/A	N/A
LNNVL	Non-reserved (cannot be functions or types)	N/A	N/A
LOAD	Non-reserved	N/A	N/A
LOAD_BAD	Non-reserved	N/A	N/A
LOAD_DISCARD	Non-reserved	N/A	N/A
LOCAL	Non-reserved	Reserved	Reserved
LOCALTIME	Reserved	Reserved	N/A
LOCALTIMESTAMP	Reserved	Reserved	N/A
LOCATION	Non-reserved	N/A	N/A
LOCATOR	N/A	Reserved	N/A
LOCK	Non-reserved	N/A	N/A
LOG	Non-reserved	N/A	N/A
LOGGING	Non-reserved	N/A	N/A
LOGIN_ANY	Non-reserved	N/A	N/A
LOGIN_FAILURE	Non-reserved	N/A	N/A
LOGIN_SUCCESS	Non-reserved	N/A	N/A
LOGOUT	Non-reserved	N/A	N/A
LOOP	Non-reserved	N/A	N/A
LOWER	N/A	Non-reserved	Reserved
M	N/A	Non-reserved	N/A
MAP	N/A	Reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
MAPPING	Non-reserved	N/A	N/A
MASKING	Non-reserved	N/A	N/A
MASTER	Non-reserved	N/A	N/A
MATCH	Non-reserved	Reserved	Reserved
MATCHED	Non-reserved	N/A	N/A
MATERIALIZED	Non-reserved	N/A	N/A
MAX	N/A	Non-reserved	Reserved
MAXEXTENTS	Non-reserved	N/A	N/A
MAXSIZE	Non-reserved	N/A	N/A
MAXTRANS	Non-reserved	N/A	N/A
MAXVALUE	Reserved	N/A	N/A
MERGE	Non-reserved	N/A	N/A
MESSAGE_LENGTH	N/A	Non-reserved	Non-reserved
MESSAGE_OCTET_LENGTH	N/A	Non-reserved	Non-reserved
MESSAGE_TEXT	N/A	Non-reserved	Non-reserved
METHOD	N/A	Non-reserved	N/A
MIN	N/A	Non-reserved	Reserved
MINEXTENTS	Non-reserved	N/A	N/A
MINUS	Reserved	N/A	N/A
MINUTE	Non-reserved	Reserved	Reserved
MINVALUE	Non-reserved	N/A	N/A
MOD	N/A	Non-reserved	N/A
MODE	Non-reserved	N/A	N/A
MODEL	Non-reserved	N/A	N/A
MODIFIES	N/A	Reserved	N/A
MODIFY	Reserved	Reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
MODULE	N/A	Reserved	Reserved
MONTH	Non-reserved	Reserved	Reserved
MORE	N/A	Non-reserved	Non-reserved
MOVE	Non-reserved	N/A	N/A
MOVEMENT	Non-reserved	N/A	N/A
MUMPS	N/A	Non-reserved	Non-reserved
NAME	Non-reserved	Non-reserved	Non-reserved
NAMES	Non-reserved	Reserved	Reserved
NATIONAL	Non-reserved (cannot be functions or types)	Reserved	Reserved
NATURAL	Reserved (functions and types allowed)	Reserved	Reserved
NCHAR	Non-reserved (cannot be functions or types)	Reserved	Reserved
NCLOB	N/A	Reserved	N/A
NEW	N/A	Reserved	N/A
NEXT	Non-reserved	Reserved	Reserved
NO	Non-reserved	Reserved	Reserved
NOCOMPRESS	Non-reserved	N/A	N/A
NOCYCLE	Reserved	N/A	N/A
NODE	Non-reserved	N/A	N/A
NOENTITYESCAPING	Non-reserved	N/A	N/A
NOLOGGING	Non-reserved	N/A	N/A
NOMAXVALUE	Non-reserved	N/A	N/A
NOMINVALUE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
NONE	Non-reserved (cannot be functions or types)	Reserved	N/A
NOT	Reserved	Reserved	Reserved
NOTHING	Non-reserved	N/A	N/A
NOTIFY	Non-reserved	N/A	N/A
NOTNULL	Reserved (functions and types allowed)	N/A	N/A
NOWAIT	Non-reserved	N/A	N/A
NULL	Reserved	Reserved	Reserved
NULLABLE	N/A	Non-reserved	Non-reserved
NULLCOLS	Non-reserved	N/A	N/A
NULLIF	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
NULLS	Non-reserved	N/A	N/A
NUMBER	Non-reserved (cannot be functions or types)	Non-reserved	Non-reserved
NUMERIC	Non-reserved (cannot be functions or types)	Reserved	Reserved
NUMSTR	Non-reserved	N/A	N/A
NVARCHAR2	Non-reserved (cannot be functions or types)	N/A	N/A
NVL	Non-reserved (cannot be functions or types)	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
NVL2	Non-reserved (cannot be functions or types)	N/A	N/A
OBJECT	Non-reserved	Reserved	N/A
OCTET_LENGTH	N/A	Non- reserved	Reserved
OF	Non-reserved	Reserved	Reserved
OFF	Non-reserved	Reserved	N/A
OFFSET	Reserved	N/A	N/A
OIDS	Non-reserved	N/A	N/A
OLD	N/A	Reserved	N/A
ON	Reserved	Reserved	Reserved
ONLY	Reserved	Reserved	Reserved
OPEN	N/A	Reserved	Reserved
OPERATION	N/A	Reserved	N/A
OPERATOR	Non-reserved	N/A	N/A
OPTIMIZATION	Non-reserved	N/A	N/A
OPTION	Non-reserved	Reserved	Reserved
OPTIONALLY	Non-reserved	N/A	N/A
OPTIONS	Non-reserved	Non- reserved	N/A
OR	Reserved	Reserved	Reserved
ORDER	Reserved	Reserved	Reserved
ORDINALITY	Non-reserved	Reserved	N/A
OUT	Non-reserved (cannot be functions or types)	Reserved	N/A
OUTER	Reserved (functions and types allowed)	Reserved	Reserved
OUTFILE	Non-reserved	N/A	N/A
OUTPUT	N/A	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
OVER	Non-reserved	N/A	N/A
OVERLAPS	Reserved (functions and types allowed)	Non-reserved	Reserved
OVERLAY	Non-reserved (cannot be functions or types)	Non-reserved	N/A
OVERRIDING	N/A	Non-reserved	N/A
OWNED	Non-reserved	N/A	N/A
OWNER	Non-reserved	N/A	N/A
PAD	N/A	Reserved	Reserved
PARAMETER	N/A	Reserved	N/A
PARAMETER_MODE	N/A	Non-reserved	N/A
PARAMETER_NAME	N/A	Non-reserved	N/A
PARAMETER_ORDINAL_POSITION	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_CATALOG	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_NAME	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_SCHEMA	N/A	Non-reserved	N/A
PARAMETERS	N/A	Reserved	N/A
PARSER	Non-reserved	N/A	N/A
PARTIAL	Non-reserved	Reserved	Reserved
PARTITION	Non-reserved	N/A	N/A
PARTITIONS	Non-reserved	N/A	N/A
PASCAL	N/A	Non-reserved	Non-reserved
PASSING	Non-reserved	N/A	N/A
PASSWORD	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
PATH	N/A	Reserved	N/A
PCTFREE	Non-reserved	N/A	N/A
PER	Non-reserved	N/A	N/A
PERCENT	Non-reserved	N/A	N/A
PERFORMANCE	Reserved	N/A	N/A
PERM	Non-reserved	N/A	N/A
PIVOT	Non-reserved	N/A	N/A
PLACING	Reserved	N/A	N/A
PLAN	Non-reserved	N/A	N/A
PLANS	Non-reserved	N/A	N/A
PLI	N/A	Non-reserved	Non-reserved
POLICY	Non-reserved	N/A	N/A
POSITION	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
POSTFIX	N/A	Reserved	N/A
PRECEDING	Non-reserved	N/A	N/A
PRECISION	Non-reserved (cannot be functions or types)	Reserved	Reserved
PREDICT	Non-reserved	N/A	N/A
PREFERRED	Non-reserved	N/A	N/A
PREFIX	Non-reserved	Reserved	N/A
PREORDER	N/A	Reserved	N/A
PREPARE	Non-reserved	Reserved	Reserved
PREPARED	Non-reserved	N/A	N/A
PRESERVE	Non-reserved	Reserved	Reserved
PRIMARY	Reserved	Reserved	Reserved
PRIOR	Non-reserved	Reserved	Reserved
PRIORER	Reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
PRIVATE	Non-reserved	N/A	N/A
PRIVILEGE	Non-reserved	N/A	N/A
PRIVILEGES	Non-reserved	Reserved	Reserved
PROCEDURAL	Non-reserved	N/A	N/A
PROCEDURE	Reserved	Reserved	Reserved
PROFILE	Non-reserved	N/A	N/A
PUBLIC	Non-reserved	Reserved	Reserved
PUBLISH	Non-reserved	N/A	N/A
PURGE	Non-reserved	N/A	N/A
QUERY	Non-reserved	N/A	N/A
QUOTE	Non-reserved	N/A	N/A
RANDOMIZED	Non-reserved	N/A	N/A
RANGE	Non-reserved	N/A	N/A
RATIO	Non-reserved	N/A	N/A
RAW	Non-reserved	N/A	N/A
READ	Non-reserved	Reserved	Reserved
READS	N/A	Reserved	N/A
REAL	Non-reserved (cannot be functions or types)	Reserved	Reserved
REASSIGN	Non-reserved	N/A	N/A
REBUILD	Non-reserved	N/A	N/A
RECHECK	Non-reserved	N/A	N/A
RECOVER	Non-reserved	N/A	N/A
RECURSIVE	Non-reserved	Reserved	N/A
RECYCLEBIN	Reserved (functions and types allowed)	N/A	N/A
REDISANYVALUE	Non-reserved	N/A	N/A
REF	Non-reserved	Reserved	N/A
REFERENCES	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
REFERENCING	N/A	Reserved	N/A
REFRESH	Non-reserved	N/A	N/A
REGEXP_LIKE	Non-reserved (cannot be functions or types)	N/A	N/A
REINDEX	Non-reserved	N/A	N/A
REJECT	Reserved	N/A	N/A
RELATIVE	Non-reserved	Reserved	Reserved
RELEASE	Non-reserved	N/A	N/A
REOPTIONS	Non-reserved	N/A	N/A
REMOTE	Non-reserved	N/A	N/A
REMOVE	Non-reserved	N/A	N/A
RENAME	Non-reserved	N/A	N/A
REPEATABLE	Non-reserved	Non- reserved	Non-reserved
REPLACE	Non-reserved	N/A	N/A
REPLICA	Non-reserved	N/A	N/A
RESET	Non-reserved	N/A	N/A
RESIZE	Non-reserved	N/A	N/A
RESTART	Non-reserved	N/A	N/A
RESTRICT	Non-reserved	Reserved	Reserved
RESULT	N/A	Reserved	N/A
RETURN	Non-reserved	Reserved	N/A
RETURNED_LENGTH	N/A	Non- reserved	Non-reserved
RETURNED_OCTET_LENGTH	N/A	Non- reserved	Non-reserved
RETURNED_SQLSTATE	N/A	Non- reserved	Non-reserved
RETURNING	Reserved	N/A	N/A
RETURNS	Non-reserved	Reserved	N/A
REUSE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
REVOKE	Non-reserved	Reserved	Reserved
RIGHT	Reserved (functions and types allowed)	Reserved	Reserved
ROLE	Non-reserved	Reserved	N/A
ROLES	Non-reserved	N/A	N/A
ROLLBACK	Non-reserved	Reserved	Reserved
ROLLUP	Non-reserved	Reserved	N/A
ROTATION	Non-reserved	N/A	N/A
ROUTINE	N/A	Reserved	N/A
ROUTINE_CATALOG	N/A	Non-reserved	N/A
ROUTINE_NAME	N/A	Non-reserved	N/A
ROUTINE_SCHEMA	N/A	Non-reserved	N/A
ROW	Non-reserved (cannot be functions or types)	Reserved	N/A
ROW_COUNT	N/A	Non-reserved	Non-reserved
ROWNUM	Reserved	N/A	N/A
ROWS	Non-reserved	Reserved	Reserved
ROWTYPE	Non-reserved	N/A	N/A
RULE	Non-reserved	N/A	N/A
SAMPLE	Non-reserved	N/A	N/A
SAVEPOINT	Non-reserved	Reserved	N/A
SCALE	N/A	Non-reserved	Non-reserved
SCHEDULE	Non-reserved	N/A	N/A
SCHEMA	Non-reserved	Reserved	Reserved
SCHEMA_NAME	N/A	Non-reserved	Non-reserved

Keyword	GaussDB	SQL:1999	SQL-92
SCOPE	N/A	Reserved	N/A
SCROLL	Non-reserved	Reserved	Reserved
SEARCH	Non-reserved	Reserved	N/A
SECOND	Non-reserved	Reserved	Reserved
SECTION	N/A	Reserved	Reserved
SECURITY	Non-reserved	Non-reserved	N/A
SELECT	Reserved	Reserved	Reserved
SELF	N/A	Non-reserved	N/A
SENSITIVE	N/A	Non-reserved	N/A
SEPARATOR	Non-reserved	N/A	N/A
SEQUENCE	Non-reserved	Reserved	N/A
SEQUENCES	Non-reserved	N/A	N/A
SERIALIZABLE	Non-reserved	Non-reserved	Non-reserved
SERVER	Non-reserved	N/A	N/A
SERVER_NAME	N/A	Non-reserved	Non-reserved
SESSION	Non-reserved	Reserved	Reserved
SESSION_USER	Reserved	Reserved	Reserved
SESSIONTIMEZONE	Reserved	N/A	N/A
SET	Non-reserved	Reserved	Reserved
SETOF	Non-reserved (cannot be functions or types)	N/A	N/A
SETS	Non-reserved	Reserved	N/A
SHARE	Non-reserved	N/A	N/A
SHIPPABLE	Non-reserved	N/A	N/A
SHOW	Non-reserved	N/A	N/A
SHRINK	Reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
SHUTDOWN	Non-reserved	N/A	N/A
SIBLINGS	Non-reserved	N/A	N/A
SIMILAR	Reserved (functions and types allowed)	Non-reserved	N/A
SIMPLE	Non-reserved	Non-reserved	N/A
SIZE	Non-reserved	Reserved	Reserved
SKIP	Non-reserved	N/A	N/A
SLAVE	Non-reserved	N/A	N/A
SLICE	Non-reserved	N/A	N/A
SMALLDATETIME	Non-reserved (cannot be functions or types)	N/A	N/A
SMALLDATETIME_FORMAT	Non-reserved	N/A	N/A
SMALLINT	Non-reserved (cannot be functions or types)	Reserved	Reserved
SNAPSHOT	Non-reserved	N/A	N/A
SOME	Reserved	Reserved	Reserved
SOURCE	Non-reserved	Non-reserved	N/A
SPACE	Non-reserved	Reserved	Reserved
SPECIFIC	N/A	Reserved	N/A
SPECIFIC_NAME	N/A	Non-reserved	N/A
SPECIFICATION	Non-reserved	N/A	N/A
SPECIFICTYPE	N/A	Reserved	N/A
SPILL	Non-reserved	N/A	N/A
SPLIT	Non-reserved	N/A	N/A
SQL	N/A	Reserved	Reserved
SQLCODE	N/A	N/A	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
SQLERROR	N/A	N/A	Reserved
SQLEXCEPTION	N/A	Reserved	N/A
SQLSTATE	N/A	Reserved	Reserved
SQLWARNING	N/A	Reserved	N/A
STABLE	Non-reserved	N/A	N/A
STANDALONE	Non-reserved	N/A	N/A
START	Non-reserved	Reserved	N/A
STARTING	Non-reserved	N/A	N/A
STARTS	Non-reserved	N/A	N/A
STATE	N/A	Reserved	N/A
STATEMENT	Non-reserved	Reserved	N/A
STATEMENT_ID	Non-reserved	N/A	N/A
STATIC	N/A	Reserved	N/A
STATISTICS	Non-reserved	N/A	N/A
STDIN	Non-reserved	N/A	N/A
STDOUT	Non-reserved	N/A	N/A
STORAGE	Non-reserved	N/A	N/A
STORE	Non-reserved	N/A	N/A
STORED	Non-reserved	N/A	N/A
STRATIFY	Non-reserved	N/A	N/A
STRICT	Non-reserved	N/A	N/A
STRIP	Non-reserved	N/A	N/A
STRUCTURE	N/A	Reserved	N/A
STYLE	N/A	Non-reserved	N/A
SUBCLASS_ORIGIN	N/A	Non-reserved	Non-reserved
SUBLIST	N/A	Non-reserved	N/A
SUBPARTITION	Non-reserved	N/A	N/A
SUBPARTITIONS	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
SUBSTRING	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
SUM	N/A	Non-reserved	Reserved
SYMMETRIC	Reserved	Non-reserved	N/A
SYNONYM	Non-reserved	N/A	N/A
SYS_REFCURSOR	Non-reserved	N/A	N/A
SYSDATE	Reserved	N/A	N/A
SYSID	Non-reserved	N/A	N/A
SYSTEM	Non-reserved	Non-reserved	N/A
SYSTEM_USER	N/A	Reserved	Reserved
TABLE	Reserved	Reserved	Reserved
TABLE_NAME	N/A	Non-reserved	Non-reserved
TABLES	Non-reserved	N/A	N/A
TABLESAMPLE	Reserved (functions and types allowed)	N/A	N/A
TABLESPACE	Non-reserved	N/A	N/A
TARGET	Non-reserved	N/A	N/A
TEMP	Non-reserved	N/A	N/A
TEMPLATE	Non-reserved	N/A	N/A
TEMPORARY	Non-reserved	Reserved	Reserved
TERMINATE	N/A	Reserved	N/A
TERMINATED	Non-reserved	N/A	N/A
TEXT	Non-reserved	N/A	N/A
THAN	Non-reserved	Reserved	N/A
THEN	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
TIME	Non-reserved (cannot be functions or types)	Reserved	Reserved
TIME_FORMAT	Non-reserved	N/A	N/A
TIMECAPSULE	Reserved (functions and types allowed)	N/A	N/A
TIMESTAMP	Non-reserved (cannot be functions or types)	Reserved	Reserved
TIMESTAMP_FORMAT	Non-reserved	N/A	N/A
TIMESTAMPDIFF	Non-reserved (cannot be functions or types)	N/A	N/A
TIMEZONE_HOUR	N/A	Reserved	Reserved
TIMEZONE_MINUTE	N/A	Reserved	Reserved
TINYINT	Non-reserved (cannot be functions or types)	N/A	N/A
TO	Reserved	Reserved	Reserved
TRAILING	Reserved	Reserved	Reserved
TRANSACTION	Non-reserved	Reserved	Reserved
TRANSACTION_ACTIVE	N/A	Non- reserved	N/A
TRANSACTIONS_COMMITTED	N/A	Non- reserved	N/A
TRANSACTIONS_ROLLED_BACK	N/A	Non- reserved	N/A
TRANSFORM	Non-reserved	Non- reserved	N/A
TRANSFORMS	N/A	Non- reserved	N/A
TRANSLATE	N/A	Non- reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
TRANSLATION	N/A	Reserved	Reserved
TREAT	Non-reserved (cannot be functions or types)	Reserved	N/A
TRIGGER	Non-reserved	Reserved	N/A
TRIGGER_CATALOG	N/A	Non- reserved	N/A
TRIGGER_NAME	N/A	Non- reserved	N/A
TRIGGER_SCHEMA	N/A	Non- reserved	N/A
TRIM	Non-reserved (cannot be functions or types)	Non- reserved	Reserved
TRUE	Reserved	Reserved	Reserved
TRUNCATE	Non-reserved	N/A	N/A
TRUSTED	Non-reserved	N/A	N/A
TSFIELD	Non-reserved	N/A	N/A
TSTAG	Non-reserved	N/A	N/A
TSTIME	Non-reserved	N/A	N/A
TYPE	Non-reserved	Non- reserved	Non-reserved
TYPES	Non-reserved	N/A	N/A
UNBOUNDED	Non-reserved	N/A	N/A
UNCOMMITTED	Non-reserved	Non- reserved	Non-reserved
UNDER	N/A	Reserved	N/A
UNENCRYPTED	Non-reserved	N/A	N/A
UNION	Reserved	Reserved	Reserved
UNIQUE	Reserved	Reserved	Reserved
UNKNOWN	Non-reserved	Reserved	Reserved
UNLIMITED	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
UNLISTEN	Non-reserved	N/A	N/A
UNLOCK	Non-reserved	N/A	N/A
UNLOGGED	Non-reserved	N/A	N/A
UNNAMED	N/A	Non-reserved	Non-reserved
UNNEST	N/A	Reserved	N/A
UNPIVOT	Non-reserved	N/A	N/A
UNTIL	Non-reserved	N/A	N/A
UNUSABLE	Non-reserved	N/A	N/A
UPDATE	Non-reserved	Reserved	Reserved
UPPER	N/A	Non-reserved	Reserved
USAGE	N/A	Reserved	Reserved
USEEOF	Non-reserved	N/A	N/A
USER	Reserved	Reserved	Reserved
USER_DEFINED_TYPE_CATALOG	N/A	Non-reserved	N/A
USER_DEFINED_TYPE_NAME	N/A	Non-reserved	N/A
USER_DEFINED_TYPE_SCHEMA	N/A	Non-reserved	N/A
USING	Reserved	Reserved	Reserved
VACUUM	Non-reserved	N/A	N/A
VALID	Non-reserved	N/A	N/A
VALIDATE	Non-reserved	N/A	N/A
VALIDATION	Non-reserved	N/A	N/A
VALIDATOR	Non-reserved	N/A	N/A
VALUE	Non-reserved	Reserved	Reserved
VALUES	Non-reserved (cannot be functions or types)	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
VARCHAR	Non-reserved (cannot be functions or types)	Reserved	Reserved
VARCHAR2	Non-reserved (cannot be functions or types)	N/A	N/A
VARIABLE	N/A	Reserved	N/A
VARIABLES	Non-reserved	N/A	N/A
VARIADIC	Reserved	N/A	N/A
VARYING	Non-reserved	Reserved	Reserved
VCGROUP	Non-reserved	N/A	N/A
VERBOSE	Reserved (functions and types allowed)	N/A	N/A
VERIFY	Reserved	N/A	N/A
VERSION	Non-reserved	N/A	N/A
VIEW	Non-reserved	Reserved	Reserved
VOLATILE	Non-reserved	N/A	N/A
WAIT	Non-reserved	N/A	N/A
WEAK	Non-reserved	N/A	N/A
WELLFORMED	Non-reserved	N/A	N/A
WHEN	Reserved	Reserved	Reserved
WHENEVER	N/A	Reserved	Reserved
WHERE	Reserved	Reserved	Reserved
WHITESPACE	Non-reserved	N/A	N/A
WINDOW	Reserved	N/A	N/A
WITH	Reserved	Reserved	Reserved
WITHIN	Non-reserved	N/A	N/A
WITHOUT	Non-reserved	Reserved	N/A
WORK	Non-reserved	Reserved	Reserved
WORKLOAD	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
WRAPPER	Non-reserved	N/A	N/A
WRITE	Non-reserved	Reserved	Reserved
XML	Non-reserved	N/A	N/A
XMLATTRIBUTES	Non-reserved (cannot be functions or types)	N/A	N/A
XMLCONCAT	Non-reserved (cannot be functions or types)	N/A	N/A
XMLELEMENT	Non-reserved (cannot be functions or types)	N/A	N/A
XML EXISTS	Non-reserved (cannot be functions or types)	N/A	N/A
XMLFOREST	Non-reserved (cannot be functions or types)	N/A	N/A
XMLPARSE	Non-reserved (cannot be functions or types)	N/A	N/A
XMLPI	Non-reserved (cannot be functions or types)	N/A	N/A
XMLROOT	Non-reserved (cannot be functions or types)	N/A	N/A
XMLSERIALIZE	Non-reserved (cannot be functions or types)	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
XMLTYPE	Non-reserved (cannot be functions or types)	N/A	N/A
YEAR	Non-reserved	Reserved	Reserved
YES	Non-reserved	N/A	N/A
ZONE	Non-reserved	Reserved	Reserved

Fields listed in the following table cannot be used as column names during table creation.

CTID	XMIN	CMIN	XMAX	CMAX
TABLEOID	XC_NODE_ID	TID	-	GS_TUPLE_UID
TABLEBUCKETID	XC_NODE_HASH	N/A	N/A	N/A

7.3 Data Type

Data type is a basic attribute of data that used to distinguish different types of data. Different data types occupy different storage space and support different operations. Data is stored in data tables in the database. Each column of a data table specifies a data type and data must be stored according to data types.

GaussDB supports implicit conversions between certain data types. For details, see [PG_CAST](#).

7.3.1 Numeric Types

[Table 7-2](#) lists all available types. For arithmetic operators and related built-in functions, see [Arithmetic Functions and Operators](#).

Table 7-2 Integer types

Name	Description	Storage Space	Range
TINYINT	Tiny integer, also called INT1	1 byte	0 ~ +255
SMALLINT	Small integer, also called INT2	2 bytes	-32,768 to +32,767

Name	Description	Storage Space	Range
INTEGER	Typical choice for integers, also called INT4	4 bytes	-2,147,483,648 to +2,147,483,647
BINARY_INTEGER	Alias of INTEGER.	4 bytes	-2,147,483,648 to +2,147,483,647
BIGINT	Big integer, also called INT8	8 bytes	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
int16	A 16-byte integer cannot be used to create tables.	16 bytes	-170,141,183,460,469,231,731,687,303,715,884,105,728 ~ +170,141,183,460,469,231,731,687,303,715,884,105,727

Example:

```
-- Create a table containing TINYINT data.
gaussdb=# CREATE TABLE int_type_t1
(
  IT_COL1 TINYINT
);

-- Insert data.
gaussdb=# INSERT INTO int_type_t1 VALUES(10);

-- View data.
gaussdb=# SELECT * FROM int_type_t1;
it_col1
-----
10
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE int_type_t1;
-- Create a table containing TINYINT, INTEGER, and BIGINT data.
gaussdb=# CREATE TABLE int_type_t2
(
  a TINYINT,
  b TINYINT,
  c INTEGER,
  d BIGINT
);

-- Insert data.
gaussdb=# INSERT INTO int_type_t2 VALUES(100, 10, 1000, 10000);

-- View data.
gaussdb=# SELECT * FROM int_type_t2;
 a | b | c | d
-----+-----+-----+-----
100 | 10 | 1000 | 10000
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE int_type_t2;
```

 NOTE

- Numbers of the TINYINT, SMALLINT, INTEGER, BIGINT, or INT16 type, that is, integers can be stored. Saving a number with a decimal in any of the data types will result in errors.
- The INTEGER type is the common choice, as it offers the best balance between range, storage size, and performance. Generally, use the SMALLINT type only if you are sure that the value range is within the SMALLINT value range. The storage speed of INTEGER is much faster. BIGINT is used only when the range of INTEGER is not large enough.

Table 7-3 Arbitrary precision types

Name	Description	Storage Space	Range
NUMERIC[(p[,s])], DECIMAL[(p[,s])]	The value range of p (precision) is [1,1000], and the value range of s (scale) is [0, <i>p</i>]. NOTE p indicates the total digits, and s indicates the decimal digit.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 131,072 digits before the decimal point, and up to 16,383 digits after the decimal point when no precision is specified.
NUMBER[(p[,s])]	Alias of the NUMERIC type.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 131,072 digits before the decimal point, and up to 16,383 digits after the decimal point when no precision is specified.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE decimal_type_t1
(
  DT_COL1 DECIMAL(10,4)
);

-- Insert data.
gaussdb=# INSERT INTO decimal_type_t1 VALUES(123456.122331);

-- Query data in the table.
gaussdb=# SELECT * FROM decimal_type_t1;
 dt_col1
-----
123456.1223
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE decimal_type_t1;
-- Create a table.
gaussdb=# CREATE TABLE numeric_type_t1
(
  NT_COL1 NUMERIC(10,4)
);
```

```
-- Insert data.
gaussdb=# INSERT INTO numeric_type_t1 VALUES(123456.12354);

-- Query data in the table.
gaussdb=# SELECT * FROM numeric_type_t1;
 nt_col1
-----
123456.1235
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE numeric_type_t1;
```

 **NOTE**

- Compared to the integer types, the arbitrary precision numbers require larger storage space and have lower storage efficiency, operation efficiency, and poorer compression ratio results. The integer type is the common choice when number types are defined. Arbitrary precision numbers are used when numbers exceed the maximum range indicated by the integers.
- When NUMERIC/DECIMAL is used for defining a column, you are advised to specify the precision (p) and scale (s) for the column.

Table 7-4 Sequence integer

Name	Description	Storage Space	Range
SMALLSERIAL	Two-byte serial integer	2 bytes.	-32,768 to +32,767.
SERIAL	Four-byte serial integer	4 bytes.	-2,147,483,648 to +2,147,483,647.
BIGSERIAL	Eight-byte serial integer	8 bytes.	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE smallserial_type_tab(a SMALLSERIAL);

-- Insert data.
gaussdb=# INSERT INTO smallserial_type_tab VALUES(default);

-- Insert data again.
gaussdb=# INSERT INTO smallserial_type_tab VALUES(default);

-- View data.
gaussdb=# SELECT * FROM smallserial_type_tab;
 a
---
 1
 2
(2 rows)

-- Create a table.
gaussdb=# CREATE TABLE serial_type_tab(b SERIAL);

-- Insert data.
```

```

gaussdb=# INSERT INTO serial_type_tab VALUES(default);
-- Insert data again.
gaussdb=# INSERT INTO serial_type_tab VALUES(default);

-- View data.
gaussdb=# SELECT * FROM serial_type_tab;
 b
----
 1
 2
(2 rows)

-- Create a table.
gaussdb=# CREATE TABLE bigserial_type_tab(c BIGSERIAL);

-- Insert data.
gaussdb=# INSERT INTO bigserial_type_tab VALUES(default);

-- Insert data again.
gaussdb=# INSERT INTO bigserial_type_tab VALUES(default);

-- View data.
gaussdb=# SELECT * FROM bigserial_type_tab;
 c
----
 1
 2
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE smallserial_type_tab;

gaussdb=# DROP TABLE serial_type_tab;

gaussdb=# DROP TABLE bigserial_type_tab;

```

 **NOTE**

SMALLSERIAL, SERIAL, and BIGSERIAL are not real types. They are concepts used for setting a unique identifier for a table. Therefore, an integer column is created and its default value plans to be read from a sequencer. A NOT NULL constraint is used to ensure NULL is not inserted. In most cases you would also want to attach an **UNIQUE** or **PRIMARY KEY** constraint to prevent duplicate values from being inserted unexpectedly, but this is not automatic. Finally, the sequencer belongs to the column. In this case, when the column or the table is deleted, the sequencer is also deleted. Currently, the **SERIAL** column can be specified only when you create a table. You cannot add the **SERIAL** column in an existing table. In addition, **SERIAL** columns cannot be created in temporary tables. Because SERIAL is not a data type, columns cannot be converted to this type.

Table 7-5 Floating-point types

Name	Description	Storage Space	Range
REAL, FLOAT4	Single precision floating points, which is not very precise.	4 bytes.	-3.402E+38 to +3.402E+38, 6-bit decimal digits.
DOUBLE PRECISION , FLOAT8	Double precision floating points, which is not very precise.	8 bytes.	-1.79E+308 to +1.79E+308, 15-bit decimal digits.

Name	Description	Storage Space	Range
FLOAT[(p)]	<p>Floating-point number, which is not very precise. The value range of precision (p) is [1,53].</p> <p>NOTE p is the precision, indicating the total number of binary bits.</p>	4 bytes or 8 bytes.	<p>REAL or DOUBLE PRECISION is selected as an internal identifier based on precision (p). If no precision is specified, DOUBLE PRECISION is used as the internal identifier.</p>
BINARY_DOUBLE	Alias for DOUBLE PRECISION, compatible with Oracle Database.	8 bytes.	-1.79E+308 to +1.79E+308, 15-bit decimal digits.
DEC[(p[,s])]	<p>The value range of p (precision) is [1,1000], and the value range of s (scale) is [0,p].</p> <p>NOTE p indicates the total digits, and s indicates the decimal digit.</p>	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Maximum 131,072 digits before the decimal point and 16,383 digits after the decimal point when the precision and scale are specified to the maximum.
INTEGER[(p[,s])]	<p>The value range of p (precision) is [1,1000], and the value range of s (scale) is [0,p].</p> <p>If the precision and scale are not specified, the precision p is 10 and the scale s is 0 by default.</p> <p>If the precision and scale are not specified, this type is mapped to INTEGER. If the precision and scale are specified, this type is mapped to NUMERIC.</p>	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	<p>Maximum 131,072 digits before the decimal point and 16,383 digits after the decimal point when the precision and scale are specified to the maximum.</p> <p>If the precision and scale are not specified, the value ranges from -2,147,483,648 to +2,147,483,647.</p>

 NOTE

- The binary floating-point data types REAL, FLOAT4, DOUBLE, DOUBLE PRECISION, FLOAT8, FLOAT[(p)], and BINARY_DOUBLE are imprecise numeric types. Their internal storage is approximate, which may result in slight differences during storage and retrieval. When using binary floating-point data types, pay attention to the following points:
 - Precise storage and calculation: If precise storage and calculation are required (such as for monetary values), use exact data types (for example, numeric) instead.
 - Complex calculation: When you perform complex calculation with imprecise data types to obtain critical data, it is important to carefully evaluate the results.
 - Floating-point number comparison: The result of comparing two floating-point numbers for equality may differ from expectations.
 - Underflow error: If a floating-point number is too close to zero, it cannot be accurately represented. As a result, an underflow error occurs.
- For the precision of the floating-point type, only the number of precision bits can be ensured when the data is directly read. When distributed computing is involved, the computation is executed on each DN and is finally aggregated to a CN. Therefore, the error may be amplified as the number of compute nodes increases.
- In [Table 7-5](#), **p** is the precision, indicating the minimum acceptable total number of integral places, and **s** indicates the decimal digit.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE float_type_t2
(
  FT_COL1 INTEGER,
  FT_COL2 FLOAT4,
  FT_COL3 FLOAT8,
  FT_COL4 FLOAT(3),
  FT_COL5 BINARY_DOUBLE,
  FT_COL6 DECIMAL(10,4),
  FT_COL7 INTEGER(6,3)
)DISTRIBUTE BY HASH ( ft_col1);

-- Insert data.
gaussdb=# INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,10.3214, 321.321, 123.123654,
123.123654);

-- View data.
gaussdb=# SELECT * FROM float_type_t2 ;
ft_col1 | ft_col2 | ft_col3 | ft_col4 | ft_col5 | ft_col6 | ft_col7
-----+-----+-----+-----+-----+-----+-----
      10 | 10.3655 | 123456.1234 | 10.3214 | 321.321 | 123.1237 | 123.124
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE float_type_t2;
```

7.3.2 Monetary Types

The monetary type stores a currency amount with fixed fractional precision.

The range shown in [Table 7-6](#) assumes there are two fractional digits. Input is accepted in a variety of formats, including integer and floating-point literals, as well as typical currency formatting, such as "\$1,000.00". Output is generally in the last format but depends on the locale.

Table 7-6 Monetary type

Name	Description	Storage Space	Range
money	Currency amount	8 bytes	-92233720368547758.08 to +92233720368547758.07

Values of the numeric, int, and bigint data types can be cast to money. Conversion from the real or double precision data type to a money value can be done by casting to numeric type first, for example:

```
gaussdb=# SELECT '12.34'::float8::numeric::money;  
money  
-----  
$12.34  
(1 row)
```

However, this is not recommended. Floating point numbers should not be used to handle money due to the potential for rounding errors.

A money value can be cast to numeric without loss of precision. Conversion to other types could potentially lose precision, and must also be done in two stages:

```
gaussdb=# SELECT '52093.89'::money::numeric::float8;  
float8  
-----  
52093.89  
(1 row)
```

When a money value is divided by another money value, the result is of the double precision type (that is, the digit type instead of the money type); the currency units cancel each other out in the division.

7.3.3 Boolean Types

Table 7-7 lists the Boolean types supported by GaussDB.

Table 7-7 Boolean type

Name	Description	Storage Space	Value
BOOLEAN	Boolean	1 byte	<ul style="list-style-type: none">• true• false• null: unknown

- Valid literal values for the "true" state include:
TRUE, 't', 'true', 'y', 'yes', '1', 'TRUE', true, on, and all non-zero values.

- Valid literal values for the "false" state include:
FALSE, 'f', 'false', 'n', 'no', '0', 0, 'FALSE', **false**, and **off**.

TRUE and **FALSE** are standard expressions, compatible with SQL statements.

Examples

Boolean values are displayed using the letters t and f.

```
-- Create a table.
gaussdb=# CREATE TABLE bool_type_t1
(
  BT_COL1 BOOLEAN,
  BT_COL2 TEXT
)DISTRIBUTE BY HASH(BT_COL2);

-- Insert data.
gaussdb=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

gaussdb=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

-- View data.
gaussdb=# SELECT * FROM bool_type_t1;
bt_col1 | bt_col2
-----+-----
t       | sic est
f       | non est
(2 rows)

gaussdb=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
bt_col1 | bt_col2
-----+-----
t       | sic est
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE bool_type_t1;
```

7.3.4 Character Types

[Table 7-8](#) lists the character data types supported by GaussDB. For string operators and related built-in functions, see [Character Processing Functions and Operators](#).

Table 7-8 Character types

Name	Description	Storage Space
CHAR(n) CHARACTER(n) NCHAR(n)	Fixed-length character string, blank padded. n indicates the string length. If it is not specified, the default precision 1 is used.	The maximum value of n is 10485760 (10 MB).

Name	Description	Storage Space
<p>VARCHAR(n) CHARACTER VARYING(n)</p>	<p>Variable-length string. In PostgreSQL-compatible mode, n indicates the string length. In other compatibility modes, n indicates the byte length.</p>	<p>The maximum value of n is 10485760 (10 MB).</p> <p>If n is not specified, the maximum storage length is 1 GB – 85 – 4 (space for storing length parameter) – Length of other columns. For example, in a table with columns (a int, b varchar, c int), the maximum length for the varchar column is 1 GB – 85 – 4 (space for storing length parameter) – 4 (length of column a int) – 4 (length of column c int) = 1,073,741,727. For details, see Example of the maximum storage length of the variable-length type.</p>

Name	Description	Storage Space
VARCHAR2(n)	Variable-length string. It is the alias of the VARCHAR(n) type. n indicates the string length.	<p>The maximum value of n is 10485760 (10 MB).</p> <p>If n is not specified, the maximum storage length is 1 GB – 85 – 4 (indicating space required for storing length of the data type) – Length of other columns. For example, in a table with columns (a int, b varchar2, c int), the maximum length for the varchar2 column is 1,073,741,727 = 1 GB – 85 – 4 (space required for storing length of the data type) – 4 (length of a int) – 4 (length of c int). For details, see Example of the maximum storage length of the variable-length type.</p>

Name	Description	Storage Space
NVARCHAR2(n)	Variable-length string. In the SQL_ASCII character set, n indicates bytes. In the non-SQL_ASCII character set, n indicates characters.	<p>The maximum value of n is 10485760 (10 MB).</p> <p>If n is not specified, the maximum storage length is 1 GB – 85 – 4 (indicating space required for storing length of the data type) – Length of other columns. For example, in a table with columns (a int, b nvarchar2, c int), the maximum length for the nvarchar2 column is 1,073,741,727 = 1 GB – 85 – 4 (space required for storing length of the data type) – 4 (length of a int) – 4 (length of c int). For details, see Example of the maximum storage length of the variable-length type.</p>

Name	Description	Storage Space
CLOB	Big text object.	<p>For Astore, the maximum size is 32 TB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 32 TB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the CLOB type may be less than 32 TB minus 1 byte.</p> <p>For Ustore, the maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the CLOB type may be less than 1 GB minus 1 byte.</p>

Name	Description	Storage Space
TEXT	Variable-length string.	The maximum storage length is 1 GB - 85 - 4 (indicating space required for storing length of the data type) - Length of other columns. For example, in a table with columns (a int, b text, c int), the maximum length for the text column is 1,073,741,727 = 1 GB - 85 - 4 (space required for storing length of the data type) - 4 (length of a int) - 4 (length of c int). For details, see Example of the maximum storage length of the variable-length type.

 NOTE

1. In addition to the restriction on the size of each column, the total size of each tuple cannot exceed 1 GB minus 85 bytes and is affected by the control header information of the column, the control header information of the tuple, and whether null fields exist in the tuple.
2. NCHAR is the alias of the bpchar type, and VARCHAR2(n) is the alias of the VARCHAR(n) type.
3. Only advanced package db_lob supports CLOBs whose size is greater than 1 GB. System functions do not support CLOBs whose size is greater than 1 GB.
4. In ORA-compatible mode, the received empty string is converted to null by default.
5. GaussDB supports a maximum of 1 GB data transfer, and the maximum size of the result string returned by the function is 1 GB.
6. For details about the character sets supported by GaussDB, see "ENCODING" in [7.15.61 CREATE DATABASE](#).

GaussDB has two other fixed-length character types, as shown in [Table 7-9](#). The name type exists only for the storage of identifiers in the internal system catalogs and is not intended for use by general users. Its length is currently defined as 64 bytes (63 usable characters plus terminator). The type "char" only uses one byte of storage. It is internally used in the system catalogs as a simplistic enumeration type.

Table 7-9 Special character types

Name	Description	Storage Space
name	Internal type for object names	64 bytes
"char"	Single-byte internal type	1 byte

Examples

- Example of data insertion where the length exceeds the specified length of the type

```

-- Create a table.
gaussdb=# CREATE TABLE char_type_t1
(
  CT_COL1 CHARACTER(4)
)DISTRIBUTE BY HASH (CT_COL1);

-- Insert data.
gaussdb=# INSERT INTO char_type_t1 VALUES ('ok');

-- Query data in the table.
gaussdb=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t1;
 ct_col1 | char_length
-----+-----
ok      |          4
(1 row)

-- Delete the table.
gaussdb=# DROP TABLE char_type_t1;

-- Create a table.
gaussdb=# CREATE TABLE char_type_t2
(
  CT_COL1 VARCHAR(5)
)DISTRIBUTE BY HASH (CT_COL1);

```

```

-- Insert data.
gaussdb=# INSERT INTO char_type_t2 VALUES ('ok');

gaussdb=# INSERT INTO char_type_t2 VALUES ('good');

-- Specify the type length. An error is reported if an inserted string exceeds this length.
gaussdb=# INSERT INTO char_type_t2 VALUES ('too long');
ERROR: value too long for type character varying(5)
CONTEXT: referenced column: ct_col1

-- Specify the type length. A string exceeding this length is truncated.
gaussdb=# INSERT INTO char_type_t2 VALUES ('too long'::varchar(5));

-- Query data.
gaussdb=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t2;
 ct_col1 | char_length
-----+-----
ok      |          2
good    |          4
too l   |          5
(3 rows)

-- Delete data.
gaussdb=# DROP TABLE char_type_t2;

```

- Example of the maximum storage length of the variable-length type

The following uses `varchar` as an example, but the same applies to `varchar2`, `nvarchar`, `nvarchar2`, and `text`.

```

-- Create a table with three columns: int, varchar, and int. According to the calculation rule, the
maximum storage length of varchar is 1,073,741,727 = 1 GB - 85 - 4 - 4 - 4.
gaussdb=# CREATE TABLE varchar_maxlength_test1 (a int, b varchar, c int) DISTRIBUTE BY HASH (a);

-- The length of varchar is 1,073,741,728, which exceeds the specified length. As a result, the
insertion fails.
gaussdb=# insert into varchar_maxlength_test1 values(1, repeat('a', 1073741728), 1);
ERROR: invalid memory alloc request size 1073741824 in tuplesort.cpp:219

-- The length of varchar is 1,073,741,727, which meets the requirement. As a result, the insertion is
successful.
gaussdb=# insert into varchar_maxlength_test1 values(1, repeat('a', 1073741727), 1);

-- Create a table with only the varchar column. According to the calculation rule, the maximum
storage length of the varchar column is 1,073,741,735 = 1 GB - 85 - 4.
gaussdb=# CREATE TABLE varchar_maxlength_test2 (a varchar) DISTRIBUTE BY HASH (a);

-- The length of varchar is 1,073,741,736, which exceeds the specified length. As a result, the
insertion fails.
insert into varchar_maxlength_test2 values(repeat('a', 1073741736));
ERROR: invalid memory alloc request size 1073741824 in tuplesort.cpp:219

-- The length of varchar is 1,073,741,735, which meets the requirement. As a result, the insertion is
successful.
insert into varchar_maxlength_test2 values(repeat('a', 1073741735));

```

7.3.5 Binary Types

[Table 7-10](#) lists the binary types supported by GaussDB.

Table 7-10 Binary types

Name	Description	Storage Space
BLOB	Binary large object (BLOB). Currently, BLOB only supports the following external access APIs: <ul style="list-style-type: none"> • DBE_LOB.GET_LENGTH • DBE_LOB.READ • DBE_LOB.WRITE • DBE_LOB.WRITE_APPEND • DBE_LOB.COPY • DBE_LOB.ERASE For details about the APIs, see DBE_LOB .	For Astore, the maximum size is 32 TB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 32 TB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the BLOB type may be less than 32 TB minus 1 byte. For Ustore, the maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the BLOB type may be less than 1 GB minus 1 byte.
RAW	Variable-length hexadecimal string.	4 bytes plus the actual binary string. The maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of this type may be less than 1 GB minus 1 byte.
BYTEA	Variable-length binary string.	4 bytes plus the actual binary string. The maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of this type may be less than 1 GB minus 1 byte.

Name	Description	Storage Space
BYTEAWITHOUTORDERWITH EQUALCOL	Variable-length binary character string (new type for the encryption feature. If the encryption type of an encrypted column is specified as deterministic encryption, the column type is BYTEAWITHOUTORDERWITH EQUALCOL). The original data type is displayed when an encrypted table is printed by running the meta command.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).
BYTEAWITHOUTORDERCOL	Variable-length binary character string (new type for the encryption feature. If the encryption type of the encrypted column is specified as random encryption, the column type is BYTEAWITHOUTORDERCOL). The original data type is displayed when the encryption table is printed by running the meta command.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).
_BYTEAWITHOUTORDERWITH EQUALCOL	Variable-length binary string, which is a new type for the encryption feature.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).
_BYTEAWITHOUTORDERCOL	Variable-length binary string, which is a new type for the encryption feature.	4 bytes plus the actual binary string. The maximum value is 1073741771 bytes (1 GB – 53 bytes).

 **NOTE**

- In addition to the size limitation on each column, the total size of each tuple is 1,073,741,823 bytes (1 GB minus 1 bytes).
- BYTEAWITHOUTORDERWITH EQUALCOL, BYTEAWITHOUTORDERCOL, _BYTEAWITHOUTORDERWITH EQUALCOL, and _BYTEAWITHOUTORDERCOL cannot be directly used to create a table.
- RAW(*n*), where *n* indicates the recommended byte length and is not used to verify the byte length of the input raw type.
- GaussDB supports a maximum of 1 GB data transfer, and the maximum size of the result string returned by the function is 1 GB.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE blob_type_t1
(
  BT_COL1 INTEGER,
  BT_COL2 BLOB,
  BT_COL3 RAW,
  BT_COL4 BYTEA
) DISTRIBUTE BY REPLICATION;

-- Insert data.
gaussdb=# INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'\xDEADBEEF');

-- Query data in the table.
gaussdb=# SELECT * FROM blob_type_t1;
bt_col1 | bt_col2 | bt_col3 | bt_col4
-----+-----+-----+-----
      10 |          | DEADBEEF | \xdeadbeef
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE blob_type_t1;
```

7.3.6 Date/Time Types

Table 7-11 lists the date/time types that can be used in GaussDB. For the operators and built-in functions of the types, see [Date and Time Processing Functions and Operators](#).

 **NOTE**

If the time format of another database is different from that of GaussDB, modify the value of the **DateStyle** parameter to keep them consistent.

Table 7-11 Date/Time types

Name	Description	Storage Space
DATE	Date. <ul style="list-style-type: none"> Minimum value: 4713-01-01BC (4713 B.C.). Maximum value: 5874897-12-31AD (5874897 A.C.) NOTE For ORA compatibility, the database treats empty strings as NULL and replaces DATE with TIMESTAMP(0) WITHOUT TIME ZONE .	4 bytes (8 bytes in ORA-compatible mode)
TIME [(p)] [WITHOUT TIME ZONE]	Time within one day. <p>p indicates the precision after the decimal point. The value ranges from 0 to 6.</p> <ul style="list-style-type: none"> Minimum value: 00:00:00. Maximum value: 24:00:00. 	8 bytes

Name	Description	Storage Space
TIME [(p)] [WITH TIME ZONE]	<p>Time within one day (with time zone).</p> <p>p indicates the precision after the decimal point. The value ranges from 0 to 6.</p> <ul style="list-style-type: none"> • Minimum value: 00:00:00+1559. • Maximum value: 24:00:00. 	12 bytes
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	<p>Date and time.</p> <p>p indicates the precision after the decimal point. The value ranges from 0 to 6.</p> <ul style="list-style-type: none"> • Minimum value: 4713-11-24BC 00:00:00.000000 (4713 B.C.). • Maximum value: 294277-01-09 AD 00:00:00.000000 (294277 A.D.). 	8 bytes
TIMESTAMP[(p)] [WITH TIME ZONE]	<p>Date and time (with time zone). TIMESTAMP is also called TIMESTAMPTZ.</p> <p>p indicates the precision after the decimal point. The value ranges from 0 to 6.</p> <ul style="list-style-type: none"> • Minimum value: 4713-11-24BC 00:00:00.000000 (4713 B.C.). • Maximum value: 294277-01-09 AD 00:00:00.000000 (294277 A.D.). <p>Time zone update: In some countries or regions, the time zone information is frequently updated due to political, economic, and war factors. Therefore, the database system needs to modify the time zone file to ensure that the time is correct.</p> <p>Currently, the GaussDB time zone type involves only timestamp with timezone. When a new time zone file takes effect, the existing data is not changed, and the new data is adjusted based on the time zone file information.</p>	8 bytes

Name	Description	Storage Space
SMALLDATETIME	<p>Date and time (without time zone). The precision level is minute. 30s to 59s are rounded into one minute.</p> <ul style="list-style-type: none"> • Minimum value: 4713-11-24BC 00:00:00.000000 (4713 B.C.). • Maximum value: 294277-01-09 AD 00:00:00.000000 (294277 A.D.). 	8 bytes
INTERVAL DAY (l) TO SECOND (p)	<p>Specifies the time interval (<i>X</i> days <i>X</i> hours <i>X</i> minutes <i>X</i> seconds).</p> <ul style="list-style-type: none"> • l: indicates the precision of days. The value ranges from 0 to 6. For compatibility, the precision functions are not supported. • p: indicates the precision of seconds. The value ranges from 0 to 6. The digit 0 at the end of a decimal number is not displayed. 	16 bytes
INTERVAL [FIELDS] [(p)]	<p>Time interval.</p> <ul style="list-style-type: none"> • FIELDS: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, HOUR TO MINUTE, HOUR TO SECOND, or MINUTE TO SECOND. • p: indicates the precision of seconds. The value ranges from 0 to 6. p takes effect only when FIELDS is set to SECOND, DAY TO SECOND, HOUR TO SECOND, or MINUTE TO SECOND. The digit 0 at the end of a decimal number is not displayed. 	12 bytes
reltime	<p>Relative time interval.</p> <ul style="list-style-type: none"> • The format is as follows: <i>X</i> years <i>X</i> mons <i>X</i> days <i>XX:XX:XX</i>. • The Julian calendar is used. It specifies that a year has 365.25 days and a month has 30 days. The relative time interval needs to be calculated based on the input value. 	4 bytes

Name	Description	Storage Space
abstime	<p>Date and time.</p> <ul style="list-style-type: none"> Format: YYYY-MM-DD hh:mm:ss +timezone. The value range is from 1901-12-13 20:45:53 GMT to 2038-01-18 23:59:59 GMT. The precision is second. 	4 bytes

 **NOTE**

1. The data of the time type automatically ignores all zeros at the end of the data when it is displayed.
2. The default value of **p** is **6**.
3. For the INTERVAL type, the date and time are stored in the int32 and double types in the system. Therefore, the value ranges of the two types are the same as those of the corresponding data type.
4. If the insertion time is out of the range, the system may not report an error, but may not ensure that the operation is normal.

 **NOTE**

If the values of **a_format_version** and **a_format_dev_version** are **10c** and **s1**, the default DATE value is determined by the following:

- Year: returned through SYSDATE
- Month: returned through SYSDATE
- Day: 01 (first day of the month)
- Hour, minute, and second: all 0

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE date_type_tab(coll date);

-- Insert data.
gaussdb=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data.
gaussdb=# SELECT * FROM date_type_tab;
 coll
-----
 2010-12-10
(1 row)

-- Delete a table.
gaussdb=# DROP TABLE date_type_tab;

-- Create a table.
gaussdb=# CREATE TABLE time_type_tab (da time without time zone ,dai time with time zone,dfgh
timestamp without time zone,dfga timestamp with time zone, vbg smalldatetime);

-- Insert data.
gaussdb=# INSERT INTO time_type_tab VALUES ('21:21:21','21:21:21 pst','2010-12-12','2013-12-11
pst','2003-04-12 04:05:06');

-- View data.
gaussdb=# SELECT * FROM time_type_tab;
```

```
da | dai | dfg | dfga | vbg
-----+-----+-----+-----+-----
21:21:21 | 21:21:21-08 | 2010-12-12 00:00:00 | 2013-12-11 16:00:00+08 | 2003-04-12 04:05:00
(1 row)

-- Delete a table.
gaussdb=# DROP TABLE time_type_tab;

-- Create a table.
gaussdb=# CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));

-- Insert data.
gaussdb=# INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);

-- View data.
gaussdb=# SELECT * FROM day_type_tab;
a | b
---+-----
1 | 3 days
(1 row)

-- Delete a table.
gaussdb=# DROP TABLE day_type_tab;

-- Create a table.
gaussdb=# CREATE TABLE year_type_tab(a int, b interval year (6));

-- Insert data.
gaussdb=# INSERT INTO year_type_tab VALUES(1,interval '2' year);

-- View data.
gaussdb=# SELECT * FROM year_type_tab;
a | b
---+-----
1 | 2 years
(1 row)

gaussdb=# SELECT TIME 'allballs';
time
-----
00:00:00
(1 row)

-- Delete a table.
gaussdb=# DROP TABLE year_type_tab;
```

Date Input

Date and time input is accepted in almost any reasonable formats, including ISO 8601, SQL-compatible, and traditional Postgres. The system allows you to customize the sequence of day, month, and year in the date input. Set the **DateStyle** parameter to **MDY** to select month-day-year interpretation, **DMY** to select day-month-year interpretation, or **YMD** to select year-month-day interpretation.

Remember that any date or time literal input needs to be enclosed with single quotation marks ('), and the syntax is as follows:

```
type [ ( p ) ] 'value'
```

The **p** that can be selected in the precision statement is an integer, indicating the number of fractional digits in the **seconds** column. [Table 7-12](#) shows the input formats of the date type.

Table 7-12 Date input formats

Example	Description
1999-01-08	ISO 8601 (recommended format). January 8, 1999 in any format.
January 8, 1999	Unambiguous in any datestyle input mode.
1/8/1999	January 8 in MDY mode and August 1 in DMY format.
1/18/1999	January 18 in MDY format, rejected in other formats.
01/02/03	<ul style="list-style-type: none"> January 2, 2003 in MDY format. February 1, 2003 in DMY format. February 3, 2001 in YMD format.
1999-Jan-08	January 8 in any format.
Jan-08-1999	January 8 in any format.
08-Jan-1999	January 8 in any format.
99-Jan-08	January 8 in YMD format, else error.
08-Jan-99	January 8, except error in YMD format.
Jan-08-99	January 8, except error in YMD format.
19990108	ISO 8601. January 8, 1999 in any format.
990108	ISO 8601. January 8, 1999 in any format.
1999.008	Year and day of year.
J2451187	Julian date.
January 8, 99 BC	Year 99 B.C.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE date_type_tab(coll date);

-- Insert data.
gaussdb=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data.
gaussdb=# SELECT * FROM date_type_tab;
      coll
-----
2010-12-10
(1 row)

-- View the date format.
gaussdb=# SHOW datestyle;
DateStyle
-----
ISO, MDY
(1 row)
```

```
-- Set the date format.
gaussdb=# SET datestyle='YMD';
SET

-- Insert data.
gaussdb=# INSERT INTO date_type_tab VALUES(date '2010-12-11');

-- View data.
gaussdb=# SELECT * FROM date_type_tab;
      coll
-----
2010-12-10
2010-12-11
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE date_type_tab;
```

Time

The time-of-day types are **TIME [(p)] [WITHOUT TIME ZONE]** and **TIME [(p)] [WITH TIME ZONE]**. **TIME** alone is equivalent to **TIME WITHOUT TIME ZONE**.

If a time zone is specified in the input for **TIME WITHOUT TIME ZONE**, it is silently ignored.

For details about the time input types, see [Table 7-13](#). For details about time zone input types, see [Table 7-14](#).

Table 7-13 Time input types

Example	Description
05:06.8	ISO 8601
4:05:06	ISO 8601
4:05	ISO 8601
040506	ISO 8601
4:05 AM	Same as 04:05. Input hours must be less than or equal to 12.
4:05 PM	Same as 16:05. Input hours must be less than or equal to 12.
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	Time zone specified by abbreviation
2003-04-12 04:05:06 America/ New_York	Time zone specified by full name

Table 7-14 Time zone input types

Example	Description
PST	Abbreviation (for Pacific Standard Time)
America/New_York	Full time zone name
-8:00	ISO-8601 offset for PST
-800	ISO-8601 offset for PST
-8	ISO-8601 offset for PST

Example:

```
gaussdb=# SELECT time '04:05:06';
time
-----
04:05:06
(1 row)

gaussdb=# SELECT time '04:05:06 PST';
time
-----
04:05:06
(1 row)

gaussdb=# SELECT time with time zone '04:05:06 PST';
timetz
-----
04:05:06-08
(1 row)
```

Special Values

The special values supported by GaussDB are converted to common date/time values when being read. For details, see [Table 7-15](#).

Table 7-15 Special values

Input String	Applicable Type	Description
epoch	date and timestamp	1970-01-01 00:00:00+00 (Unix system time zero)
infinity	timestamp	Later than any other timestamps
-infinity	timestamp	Earlier than any other timestamps
now	date, time, and timestamp	Start time of the current transaction
today	date and timestamp	Midnight today
tomorrow	date and timestamp	Midnight tomorrow
yesterday	date and timestamp	Midnight yesterday

Input String	Applicable Type	Description
allballs	time	00:00:00.00 UTC

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE realtime_type_special(col1 varchar(20), col2 date, col3 timestamp, col4 time);

-- Insert data.
gaussdb=# INSERT INTO realtime_type_special VALUES('epoch', 'epoch', 'epoch', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('now', 'now', 'now', 'now');
gaussdb=# INSERT INTO realtime_type_special VALUES('today', 'today', 'today', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('tomorrow', 'tomorrow', 'tomorrow', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('yesterday', 'yesterday', 'yesterday', NULL);

-- View data.
gaussdb=# SELECT * FROM realtime_type_special;
 col1 | col2 | col3 | col4
-----+-----+-----+-----
epoch | 1970-01-01 00:00:00 | 1970-01-01 00:00:00 |
now | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00 |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 < 'infinity';
 col1 | col2 | col3 | col4
-----+-----+-----+-----
epoch | 1970-01-01 00:00:00 | 1970-01-01 00:00:00 |
now | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00 |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > '-infinity';
 col1 | col2 | col3 | col4
-----+-----+-----+-----
epoch | 1970-01-01 00:00:00 | 1970-01-01 00:00:00 |
now | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00 |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > 'now';
 col1 | col2 | col3 | col4
-----+-----+-----+-----
tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 = 'today';
 col1 | col2 | col3 | col4
-----+-----+-----+-----
today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
(1 row)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 = 'tomorrow';
 col1 | col2 | col3 | col4
-----+-----+-----+-----
tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)
```

```
gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > 'yesterday';
 col1 | col2 | col3 | col4
-----+-----+-----+-----
now | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(3 rows)

-- Delete a table.
gaussdb=# DROP TABLE realtime_type_special;
```

Interval Input

The input of **reltime** can be any valid interval in text format. It can be a number (negative numbers and decimals are also allowed) or a specific time, which must be in SQL standard format, ISO-8601 format, or POSTGRES format. In addition, the text input needs to be enclosed with single quotation marks (").

For details about interval input, see [Table 7-16](#).

Table 7-16 Interval input

Input	Output	Description
60	2 mons	Numbers are used to indicate intervals. The default unit is day. Decimals and negative numbers are allowed. Particularly, a negative interval syntactically means how long before.
31.25	1 mons 1 days 06:00:00	
-365	-12 mons -5 days	
1 years 1 mons 8 days 12:00:00	1 years 1 mons 8 days 12:00:00	Intervals are in POSTGRES format. They can contain both positive and negative numbers and are case-insensitive. Output is a simplified POSTGRES interval converted from the input.
-13 months -10 hours	-1 years -25 days -04:00:00	
-2 YEARS +5 MONTHS 10 DAYS	-1 years -6 mons -25 days -06:00:00	
P-1.1Y10M	-3 mons -5 days -06:00:00	Intervals are in ISO-8601 format. They can contain both positive and negative numbers and are case-insensitive. Output is a simplified POSTGRES interval converted from the input.
-12H	-12:00:00	

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

-- Insert data.
gaussdb=# INSERT INTO reltime_type_tab VALUES ('90', '90');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('-366', '-366');
```

```

gaussdb=# INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS', '-2 YEARS +5 MONTHS 10 DAYS');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 12:00:00');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');

-- View data.
gaussdb=# SELECT * FROM reltime_type_tab;
   col1          |          col2
-----+-----
    90           |    3 mons
   -366          | -1 years -18:00:00
   1975.25       | 5 years 4 mons 29 days
-2 YEARS +5 MONTHS 10 DAYS | -1 years -6 mons -25 days -06:00:00
30 DAYS 12:00:00 | 1 mon 12:00:00
P-1.1Y10M       | -3 mons -5 days -06:00:00
(6 rows)

-- Drop the table.
gaussdb=# DROP TABLE reltime_type_tab;

```

7.3.7 Geometric Types

Table 7-17 lists the geometric types that can be used in GaussDB. The most fundamental type, the point, forms the basis for all of the other types.

Table 7-17 Geometric types

Name	Storage Space	Description	Representation
point	16 bytes	Point on a plane	(x,y)
lseg	32 bytes	Finite line segment	((x1,y1),(x2,y2))
box	32 bytes	Rectangle	((x1,y1),(x2,y2))
path	16 + 16 <i>n</i> bytes	Closed path (similar to polygon)	((x1,y1),...)
path	16 + 16 <i>n</i> bytes	Open path	[(x1,y1),...]
polygon	40 + 16 <i>n</i> bytes	Polygon (similar to closed paths)	((x1,y1),...)
circle	24 bytes	Circle	<(x,y),r> (center point and radius)

A rich set of functions and operators is available in GaussDB to perform various geometric operations, such as scaling, translation, rotation, and determining intersections. For details, see [Geometric Functions and Operators](#).

Points

Points are the fundamental two-dimensional building block for geometric types. Values of the **point** type are specified using either of the following syntax:

```
( x , y )  
x , y
```

x and **y** are the respective coordinates, as floating-point numbers. The value type of the points is float8.

Points are output using the first syntax.

Example:

```
gaussdb=# select point(1.1, 2.2);  
point  
-----  
(1.1,2.2)  
(1 row)
```

Line Segments

Line segments (**lseg**) are represented by pairs of points. Values of the **lseg** type are specified using any of the following syntaxes:

```
[ ( x1 , y1 ) , ( x2 , y2 ) ]  
( ( x1 , y1 ) , ( x2 , y2 ) )  
( x1 , y1 ) , ( x2 , y2 )  
x1 , y1 , x2 , y2
```

(x1,y1) and **(x2,y2)** are the end points of the line segment. The value type of the points is float8.

Line segments are output using the first syntax.

Example:

```
gaussdb=# select lseg(point(1.1, 2.2), point(3.3, 4.4));  
lseg  
-----  
[(1.1,2.2),(3.3,4.4)]  
(1 row)
```

Rectangles

Rectangles are represented by pairs of points that are opposite corners of a rectangle. Values of the **box** type are specified using any of the following syntaxes:

```
(( x1 , y1 ) , ( x2 , y2 ) )  
( x1 , y1 ) , ( x2 , y2 )  
x1 , y1 , x2 , y2
```

(x1,y1) and **(x2,y2)** are any two opposite corners of the rectangle. The value type of the points is float8.

Rectangles are output using the second syntax.

Any two opposite corners can be supplied on input, but in this order, the values will be reordered as needed to store the upper right and lower left corners.

Example:

```
gaussdb=# SELECT box(point(1.1, 2.2), point(3.3, 4.4));  
box  
-----  
(3.3,4.4),(1.1,2.2)  
(1 row)
```

Paths

Paths are represented by lists of connected points. Paths can be open, where the first and last points in the list are considered not connected, or closed, where the first and last points are considered connected.

Values of the **path** type are specified using any of the following syntaxes:

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]  
( ( x1 , y1 ) , ... , ( xn , yn ) )  
( x1 , y1 ) , ... , ( xn , yn )  
( x1 , y1 , ... , xn , yn )  
x1 , y1 , ... , xn , yn
```

The points are the end points of the line segments comprising the path. The value type of the points is float8. Square brackets ([]) indicate an open path, while parentheses (()) indicate a closed path. When the outermost parentheses are omitted, as in the third through fifth syntax, a closed path is assumed.

Paths are output using the first or second syntax.

Example:

```
gaussdb=# select path(polygon '((0,0),(1,1),(2,0)'));  
path  
-----  
((0,0),(1,1),(2,0))  
(1 row)
```

Polygons

Polygons are represented by lists of points (the vertexes of the polygon). Polygons are very similar to closed paths, but are stored differently and have their own set of support functions.

Values of the **polygon** type are specified using any of the following syntaxes:

```
(( x1 , y1 ) , ... , ( xn , yn ) )  
( x1 , y1 ) , ... , ( xn , yn )  
( x1 , y1 , ... , xn , yn )  
x1 , y1 , ... , xn , yn
```

A point indicates the vertex of a polygon. The value type of a point is float8.

Polygons are output using the first syntax.

Example:

```
gaussdb=# select polygon(box '((0,0),(1,1)'));  
polygon  
-----  
((0,0),(0,1),(1,1),(1,0))  
(1 row)
```

Circles

Circles are represented by a center point and radius. Values of the **circle** type are specified using the following syntaxes:

```
< ( x , y ) , r >  
(( x , y ) , r )  
( x , y ) , r  
x , y , r
```

(x,y) is the center point and **r** is the radius of the circle. The value type of the points is float8.

Circles are output using the first syntax.

Example:

```
gaussdb=# select circle(point(0,0),1);
 circle
-----
<(0,0),1>
(1 row)
```

7.3.8 Network Address Types

GaussDB offers data types to store IPv4 and MAC addresses.

It is better to use these types instead of plain-text types to store network addresses, because these types offer input error checking and specialized operators and functions (see [Network Address Functions and Operators](#)).

Table 7-18 Network address types

Name	Storage Space	Description
cidr	7 bytes	IPv4 networks
inet	7 bytes	IPv4 hosts and networks
macaddr	6 bytes	MAC address

cidr

The classless inter-domain routing (cidr) type holds an IPv4 network address. The format for specifying networks is **address/y** where **address** is the network represented as an IPv4 address, and **y** is the number of bits in the netmask. If **y** is omitted, it is calculated using assumptions from the older classful network numbering system, except it will be at least large enough to include all of the octets written in the input. For details, see [Table 7-19](#).

Table 7-19 cidr type input examples

cidr Input	cidr Output	abbrev(cidr)
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
10.1.2	10.1.2.0/24	10.1.2/24

cidr Input	cidr Output	abbrev(cidr)
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32

Example:

```
gaussdb=# CREATE TABLE cidr_test(id int, c cidr);
CREATE TABLE
gaussdb=# INSERT INTO cidr_test VALUES (1, '192.168.100.128/25');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (2, '192.168/24');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (3, '192.168/25');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (4, '192.168.1');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (5, '192.168');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (6, '10.1.2');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (7, '10.1');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (8, '10');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (9, '2001:4f8:3:ba::/64');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (10, '2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (11, '::ffff:127.0.0.0/120');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (12, '::ffff:127.0.0.0/128');
INSERT 0 1
gaussdb=# SELECT * FROM cidr_test ORDER BY id;
 id |          c
-----+-----
  1 | 192.168.100.128/25
  2 | 192.168.0.0/24
  3 | 192.168.0.0/25
  4 | 192.168.1.0/24
  5 | 192.168.0.0/24
  6 | 10.1.2.0/24
  7 | 10.1.0.0/16
  8 | 10.0.0.0/8
  9 | 2001:4f8:3:ba::/64
 10 | 2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128
 11 | ::ffff:127.0.0.0/120
 12 | ::ffff:127.0.0.0/128
(12 rows)

gaussdb=# DROP TABLE cidr_test;
DROP TABLE
```

inet

The inet type holds an IPv4 host address, and optionally its subnet, all in one field. The subnet is represented by the number of network address bits present in the host address (the "netmask"). If the netmask is 32 and the address is an IPv4 address, then the value does not indicate a subnet, only a single host.

The input format for this type is **address/y** where **address** is an IPv4 address and **y** is the number of bits in the netmask. If the **/y** portion is omitted, the netmask is 32 for an IPv4 address, and the value represents just a single host. On display, the **/y** portion is suppressed if the netmask specifies a single host.

The essential difference between the **inet** and **cidr** data types is that **inet** accepts values with nonzero bits to the right of the netmask, whereas **cidr** does not.

Example:

```
gaussdb=# CREATE TABLE inet_test(id int, i inet);
CREATE TABLE
gaussdb=# INSERT INTO inet_test VALUES (1, '192.168.100.128/25');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (2, '192.168.100.128');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (3, '192.168.1.0/24');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (4, '192.168.1.0/25');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (5, '192.168.1.255/24');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (6, '192.168.1.255/25');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (7, '10.1.2.3/8');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (8, '11.1.2.3/16');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (9, '12.1.2.3/24');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (10, '13.1.2.3/32');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (11, '2001:4f8:3:ba::/64');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (12, '2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (13, '::ffff:127.0.0.0/120');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (14, '::ffff:127.0.0.0/128');
INSERT 0 1
gaussdb=# SELECT * FROM inet_test ORDER BY id;
 id |          i
-----+-----
  1 | 192.168.100.128/25
  2 | 192.168.100.128
  3 | 192.168.1.0/24
  4 | 192.168.1.0/25
  5 | 192.168.1.255/24
  6 | 192.168.1.255/25
  7 | 10.1.2.3/8
  8 | 11.1.2.3/16
  9 | 12.1.2.3/24
 10 | 13.1.2.3
 11 | 2001:4f8:3:ba::/64
 12 | 2001:4f8:3:ba:2e0:81ff:fe22:d1f1
 13 | ::ffff:127.0.0.0/120
 14 | ::ffff:127.0.0.0
(14 rows)

gaussdb=# DROP TABLE inet_test;
DROP TABLE
```

macaddr

The **macaddr** type stores MAC addresses, known for example from Ethernet card hardware addresses (although MAC addresses are used for other purposes as well). Input is accepted in the following formats:

```
'08:00:2b:01:02:03'  
'08-00-2b-01-02-03'  
'08002b:010203'  
'08002b-010203'  
'0800.2b01.0203'  
'08002b010203'
```

These examples would all specify the same address. Upper and lower cases are accepted for the digits a through f. Output is always in the first of the forms shown.

Example:

```
gaussdb=# CREATE TABLE macaddr_test(id int, m macaddr);  
CREATE TABLE  
gaussdb=# INSERT INTO macaddr_test VALUES (1, '08:00:2b:01:02:03');  
INSERT 0 1  
gaussdb=# INSERT INTO macaddr_test VALUES (2, '08-00-2b-01-02-03');  
INSERT 0 1  
gaussdb=# INSERT INTO macaddr_test VALUES (3, '08002b:010203');  
INSERT 0 1  
gaussdb=# INSERT INTO macaddr_test VALUES (4, '08002b-010203');  
INSERT 0 1  
gaussdb=# INSERT INTO macaddr_test VALUES (5, '0800.2b01.0203');  
INSERT 0 1  
gaussdb=# INSERT INTO macaddr_test VALUES (6, '08002b010203');  
INSERT 0 1  
gaussdb=# SELECT * FROM macaddr_test ORDER BY id;  
 id |      m  
----+-----  
  1 | 08:00:2b:01:02:03  
  2 | 08:00:2b:01:02:03  
  3 | 08:00:2b:01:02:03  
  4 | 08:00:2b:01:02:03  
  5 | 08:00:2b:01:02:03  
  6 | 08:00:2b:01:02:03  
(6 rows)  
  
gaussdb=# DROP TABLE macaddr_test;  
DROP TABLE
```

7.3.9 Bit String Types

Bit strings are strings of 1's and 0's. They can be used to store bit masks.

GaussDB supports two bit string types: `bit(n)` and `bit varying(n)`. Here, n is a positive integer. The maximum value of n is **83886080**, which is equivalent to 10 MB.

The bit type data must match the length n exactly. An error will be reported if the data length of the storage is not matched. Data of the bit varying type is of variable length up to the maximum length n . Longer strings will be rejected.

Writing **bit** without a length is equivalent to **bit(1)**, while **bit varying** without a length specification means unlimited length.

NOTE

- If one explicitly casts a bit-string value to **bit(n)**, it will be truncated or zero-padded on the right to be exactly n bits, without raising an error.
- Similarly, if one explicitly casts a bit-string value to `bit varying(n)` and the value is more than n bits, it will be truncated on the right.
- When the ADMS platform driver version 8.1.3-200 or earlier is used, use **::bit varying** to convert the bit type. Otherwise, an error may occur.

```
-- Create a table.
gaussdb=# CREATE TABLE bit_type_t1
(
  BT_COL1 INTEGER,
  BT_COL2 BIT(3),
  BT_COL3 BIT VARYING(5)
) DISTRIBUTE BY REPLICATION;

-- Insert data.
gaussdb=# INSERT INTO bit_type_t1 VALUES(1, B'101', B'00');

-- Specify the type length. An error is reported if an inserted string exceeds this length.
gaussdb=# INSERT INTO bit_type_t1 VALUES(2, B'10', B'101');
ERROR: bit string length 2 does not match type bit(3)
CONTEXT: referenced column: bt_col2

-- Specify the type length. Data is converted if it exceeds this length.
gaussdb=# INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');

-- View data.
gaussdb=# SELECT * FROM bit_type_t1;
 bt_col1 | bt_col2 | bt_col3
-----+-----+-----
      1 | 101    | 00
      2 | 100    | 101
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE bit_type_t1;
```

7.3.10 UUID Type

The data type UUID stores universally unique identifiers (UUID) as defined by RFC 4122, ISO/IEF 9834-8:2005, and related standards. This identifier is a 128-bit quantity that is generated by an algorithm chosen to make it very unlikely that the same identifier will be generated by anyone else in the known universe using the same algorithm.

Therefore, for distributed systems, these identifiers provide a better uniqueness guarantee than sequence generators, which are only unique within a single database.

A UUID is written as a sequence of lower-case hexadecimal digits, in several groups separated by hyphens, specifically a group of 8 digits followed by three groups of 4 digits followed by a group of 12 digits, for a total of 32 digits representing the 128 bits. An example of a UUID in this standard form is:

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

GaussDB also accepts the following alternative forms for input: use of upper-case letters and digits, the standard format surrounded by braces, omitting some or all hyphens, and adding a hyphen after any group of four digits. An example is provided as follows:

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
```

Output is always in the standard form.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE uuid_test(id int, test uuid) DISTRIBUTE BY HASH(test);
```

```
-- Insert data in the example format.
gaussdb=# INSERT INTO uuid_test VALUES(1, 'A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11');
gaussdb=# INSERT INTO uuid_test VALUES(2, '{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}');
gaussdb=# INSERT INTO uuid_test VALUES(3, 'a0eebc999c0b4ef8bb6d6bb9bd380a11');
gaussdb=# INSERT INTO uuid_test VALUES(4, 'a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11');

-- View data and output the data in the standard format.
gaussdb=# SELECT * FROM uuid_test;
 id |          test
-----+-----
  1 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  2 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  3 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  4 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
(4 rows)
```

7.3.11 JSON/JSONB Types

JavaScript Object Notation (JSON) data can be a single scalar, an array, or a key-value pair object. The array and object can be called a container:

- Scalar: a number, Boolean, string, or null
- Array: defined in a pair of square brackets ([]), in which elements can be any type of JSON data, and are not necessarily of the same type.
- Object: defined in a pair of braces ({}), in which objects are stored in the format of key:value. Each key must be a string enclosed by a pair of double quotation marks (""), and its value can be any type of JSON data. In case of duplicate keys, the last key value will be used.

GaussDB offers two types for storing JSON data: JSON and JSONB. The JSON data type stores a complete copy of the input, retaining the entered spaces, duplicate keys, and sequence, while the JSONB data type stores data in a decomposed binary form, removing semantic-irrelevant details and duplicate keys, and sorting key-values. Therefore, the JSONB data does not need to be parsed.

It can be found that both are of JSON type, and the same strings can be entered as input. The main difference between them is the efficiency. Because JSON data type stores an exact copy of the input text, the data must be parsed on every execution. In contrast, JSONB data is stored in a decomposed binary form and can be processed faster, though this makes it slightly slower to input due to the conversion mechanism. In addition, because the JSONB data type is normalized, it supports more operations, for example, comparing sizes according to a specific rule. JSONB also supports indexing, which is a significant advantage.

Input Format

An input must be a JSON-compliant string, which is enclosed in single quotation marks (').

null (null-json): The value can only be **null** in lowercase.

```
gaussdb=# SELECT 'null':json; -- suc
 json
-----
 null
(1 row)

gaussdb=# SELECT 'NULL':jsonb; -- err
ERROR: invalid input syntax for type jsonb
```

Number (num-json): The value can be a positive or negative integer, decimal fraction, or 0. The scientific notation is supported.

```
gaussdb=# SELECT '1'::json;
json
-----
1
(1 row)

gaussdb=# SELECT '-1.5'::json;
json
-----
-1.5
(1 row)

gaussdb=# SELECT '-1.5e-5'::jsonb, '-1.5e+2'::jsonb;
jsonb | jsonb
-----+-----
-0.000015 | -150
(1 row)

gaussdb=# SELECT '001'::json, '+15'::json, 'NaN'::json; -- Redundant leading zeros, plus signs (+), NaN, and
infinity are not supported.
ERROR: invalid input syntax for type json
```

Boolean (bool-json): The value can only be **true** or **false** in lowercase.

```
gaussdb=# SELECT 'true'::json;
json
-----
true
(1 row)

gaussdb=# SELECT 'false'::jsonb;
jsonb
-----
false
(1 row)
```

String (str-json): The value must be a string enclosed in double quotation marks ("").

```
gaussdb=# SELECT '"a"'::json;
json
-----
"a"
(1 row)

gaussdb=# SELECT '"abc"'::jsonb;
jsonb
-----
"abc"
(1 row)
```

Object (object-json): The value is enclosed in braces ({}). The key must be a JSON-compliant string, and the value can be any valid JSON string.

```
gaussdb=# SELECT '{}'::json;
json
-----
{}
(1 row)

gaussdb=# SELECT '{"a": 1, "b": {"a": 2, "b": null}}'::json;
json
-----
{"a": 1, "b": {"a": 2, "b": null}}
(1 row)
```

```
gaussdb=# SELECT '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}':jsonb;
          jsonb
-----
{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}
(1 row)
```

 CAUTION

- Note that **'null':json** and **null:json** are different, which are similar to the strings **str="null"** and **str=null**.
- For numbers, when scientific notation is used, JSONB expands them, while JSON stores an exact copy of the input text.

JSONB Advanced Features

- Precautions
 - It cannot be used as a partition key.
 - Foreign tables are not supported.

The main difference between JSON and JSONB lies in the storage mode. JSONB stores parsed binary data, which reflects the JSON hierarchy and facilitates direct access. Therefore, JSONB has many advanced features that JSON does not have.

- Format normalization
 - After the input object-json string is parsed into JSONB binary, semantically irrelevant details are naturally discarded, for example, spaces:


```
gaussdb=# SELECT ' [1, " a ", {"a" :1  } ] ':jsonb;
          jsonb
-----
[1, " a ", {"a": 1}]
(1 row)
```
 - For object-json, duplicate key-values are deleted and only the last key-value is retained. For example:


```
gaussdb=# SELECT '{"a" : 1, "a" : 2}':jsonb;
          jsonb
-----
{"a": 2}
(1 row)
```
 - For object-json, key-values will be re-sorted. The sorting rule is as follows:
 1. Longer key-values are sorted last.
 2. If the key-values are of the same length, the key-values with a larger ASCII code are sorted after the key-values with a smaller ASCII code:

```
gaussdb=# SELECT '{"aa" : 1, "b" : 2, "a" : 3}':jsonb;
          jsonb
-----
{"a": 3, "b": 2, "aa": 1}
(1 row)
```
- Size comparison

Format normalization ensures that only one form of JSONB data exists in the same semantics. Therefore, sizes may be compared according to a specific rule.

 - First, type comparison: **object-jsonb > array-jsonb > bool-jsonb > num-jsonb > str-jsonb > null-jsonb**

- Content comparison if the data type is the same:
 - str-jsonb: The default text sorting rule of the database is used for comparison. A positive value indicates greater than, a negative value indicates less than, and **0** indicates equal.
 - num-jsonb: numeric comparison
 - bool-jsonb: **true > false**
 - **array-jsonb**: long elements > short elements. If the lengths are the same, compare each element in sequence.
 - **object-jsonb**: If the length of a key-value pair is longer than that of a short key-value pair, the key is compared first, and then the value is compared.

 CAUTION

For comparison within the object-jsonb type, the final result after format sorting is used for comparison. Therefore, the comparison result may not be intuitive compared with the direct input.

- Creating indexes and primary keys
 - B-tree index
B-tree indexes and primary keys can be created for the JSONB type.
- Inclusion and existence
Querying whether a JSON contains some elements or whether some elements exist in a JSON is an important capability of JSONB.

```
-- Simple scalar/primitive values contain only the identical value.
gaussdb=# SELECT "'foo'::jsonb @> 'foo'::jsonb;
?column?
-----
t
(1 row)

-- The array on the left contains the character string on the right.
gaussdb=# SELECT '[1, "aa", 3]::jsonb ? 'aa';
?column?
-----
t
(1 row)

-- The array on the left contains all elements of the array on the right, regardless of the sequence and
repetition.
gaussdb=# SELECT '[1, 2, 3]::jsonb @> '[1, 3, 1]::jsonb;
?column?
-----
t
(1 row)

-- The object-json on the left contains all key-value pairs of object-json on the right.
gaussdb=# SELECT '{"product": "PostgreSQL", "version": 9.4, "jsonb":true}'::jsonb @>
 '{"version":9.4}'::jsonb;
?column?
-----
t
(1 row)
```

```
-- The array on the left does not contain all elements of the array on the right. The three elements on
the left are 1, 2, and [1,3], but the elements on the right are 1 and 3.
gaussdb=# SELECT '[1, 2, [1, 3]]::jsonb @> '[1, 3]]::jsonb;
?column?
-----
f
(1 row)

gaussdb=# SELECT '{"foo": {"bar": "baz"}}::jsonb @> '{"bar": "baz"}::jsonb;
?column?
-----
f
(1 row)
```

For details about the operators, see [JSON/JSONB Functions and Operators](#).

- Functions and operators

For details about the functions and operators supported by the JSON/JSONB type, see [JSON/JSONB Functions and Operators](#).

7.3.12 HLL Types

HyperLoglog (HLL) is an approximation algorithm for efficiently counting the number of distinct values in a dataset. It features faster computing and lower space usage. You only need to store HLL data structures, instead of data sets. When new data is added to a dataset, make hash calculation on the data and insert the result to an HLL. Then, you can obtain the final result based on the HLL.

[Table 7-20](#) compares HLL with other algorithms.

Table 7-20 Comparison between HLL and other algorithms

Item	Sorting Algorithm	Hash Algorithm	HLL
Time complexity	O(nlogn)	O(n)	O(n)
Space complexity	O(n)	O(n)	log(logn)
Error rate	0	0	≈0.8%
Storage space requirement	Size of original data	Size of original data	The maximum size is 16 KB by default.

HLL has advantages over others in the computing speed and storage space requirement. In terms of time complexity, the sorting algorithm needs O(nlogn) time for sorting, and the hash algorithm and HLL need O(n) time for full table scanning. In terms of storage space requirements, the sorting algorithm and hash algorithm need to store raw data before collecting statistics, whereas the HLL algorithm needs to store only the HLL data structures rather than the raw data, and thereby occupying a fixed space of about 16 KB.

NOTICE

- In the current default specifications, the maximum number of distinct values that can be calculated is about $1.1e + 15$, and the error rate is 0.8%. If the calculation result exceeds the maximum, the error rate of the calculation result will increase, or the calculation will fail and an error will be reported.
- When using this feature for the first time, you need to evaluate the distinct values of the service, properly select configuration parameters, and perform verification to ensure that the accuracy meets requirements.
 - By default, the distinct value is $1.1e + 15$. If the distinct value is NaN, you need to adjust `log2m` or use another algorithm to calculate the distinct value.
 - The hash algorithm has an extremely low probability of collision. However, you are still advised to select 2 or 3 hash seeds for verification when using the hash algorithm for the first time. If there is only a small difference between the distinct values, you can select any one of the seeds as the hash seed.

Table 7-21 describes main HLL data structures.

Table 7-21 Main HLL data structures

Data Type	Description
hll	The HLL header is a 27-byte field. By default, the data length ranges from 0 KB to 16 KB. The distinct value can be obtained.

When you create an HLL data type, 0 to 4 input parameters are supported. The parameter meanings and specifications are the same as those of the `hll_empty` function. The first parameter is **log2m**, indicating the logarithm of the number of buckets, and its value ranges from 10 to 16. The second parameter is **log2explicit**, indicating the threshold in explicit mode, and its value ranges from 0 to 12. The third parameter is **log2sparse**, indicating the threshold of the Sparse mode, and its value ranges from 0 to 14. The fourth parameter is **duplicatecheck**, indicating whether to enable duplicatecheck, and its value ranges from 0 to 1. When the input parameter is set to **-1**, the default value of the HLL parameter is used. You can run the `\d` or `\d+` command to view the parameters of the HLL type.

 **NOTE**

When the HLL data type is created, the result varies depending on the input parameter behavior:

- When creating an HLL type, do not set the input parameter or set it to **-1**. Use the default value of the corresponding HLL parameter.
- If a valid value is set for the input parameter, the corresponding HLL parameter uses the input value.
- If the input value is invalid, an error is reported when the HLL type is created.

```
-- Create an HLL table without specifying input parameters.
gaussdb=# create table t1 (id integer, set hll);
```

```

gaussdb=# \d t1
      Table "public.t1"
  Column | Type  | Modifiers
-----+-----+-----
 id      | integer |
 set     | hll    |

-- Create an HLL table, specify the first two input parameters, and use the default values for the last two
input parameters.
gaussdb=# create table t2 (id integer, set hll(12,4));
gaussdb=# \d t2
      Table "public.t2"
  Column | Type      | Modifiers
-----+-----+-----
 id      | integer   |
 set     | hll(12,4,12,0) |

-- Create an HLL table, specify the third input parameter, and use default values for other parameters.
gaussdb=# create table t3(id int, set hll(-1,-1,8,-1));
gaussdb=# \d t3
      Table "public.t3"
  Column | Type      | Modifiers
-----+-----+-----
 id      | integer   |
 set     | hll(14,10,8,0) |

-- When a user creates an HLL table and specifies an invalid input parameter, an error is reported.
gaussdb=# create table t4(id int, set hll(5,-1));
ERROR:  log2m = 5 is out of range, it should be in range 10 to 16, or set -1 as default

-- Drop the created HLL table.
gaussdb=# drop table t1,t2,t3;
DROP TABLE

```

NOTE

When inserting an HLL object to an HLL table, ensure that the parameters of the HLL type are the same as those of the inserted object. Otherwise, an error is reported.

```

-- Create an HLL table.
gaussdb=# create table t1(id integer, set hll(14));

-- Insert an HLL object to a table. The insertion succeeds because parameter types are consistent.
gaussdb=# insert into t1 values (1, hll_empty(14,-1));

-- Insert an HLL object to a table. The insertion fails because parameter types are inconsistent.
gaussdb=# insert into t1(id, set) values (1, hll_empty(14,5));
ERROR:  log2explicit does not match: source is 5 and dest is 10

-- Delete the table.
gaussdb=# drop table t1;

```

The following describes HLL application scenarios:

- Scenario 1: "Hello World"

The following example shows how to use the HLL data type:

```

-- Create an HLL table.
gaussdb=# create table helloworld (id integer, set hll);

-- Insert an empty HLL to the table.
gaussdb=# insert into helloworld(id, set) values (1, hll_empty());

-- Add a hashed integer to the HLL.
gaussdb=# update helloworld set set = hll_add(set, hll_hash_integer(12345)) where id = 1;

-- Add a hashed string to the HLL.
gaussdb=# update helloworld set set = hll_add(set, hll_hash_text('hello world')) where id = 1;

-- Obtain the number of distinct values of the HLL.

```

```
gaussdb=# select hll_cardinality(set) from helloworld where id = 1;
 hll_cardinality
-----
                2
(1 row)

-- Drop the table.
gaussdb=# drop table helloworld;
```

- Scenario 2: Collect statistics about website visitors.

The following example shows how an HLL collects statistics on the number of users visiting a website within a period of time:

```
-- Create a raw data table to show that a user has visited the website at a certain time:
gaussdb=# create table facts (
    date      date,
    user_id   integer
);

-- Create a raw data table to show that a user has visited the website at a certain time:
gaussdb=# insert into facts values ('2019-02-20', generate_series(1,100));
gaussdb=# insert into facts values ('2019-02-21', generate_series(1,200));
gaussdb=# insert into facts values ('2019-02-22', generate_series(1,300));
gaussdb=# insert into facts values ('2019-02-23', generate_series(1,400));
gaussdb=# insert into facts values ('2019-02-24', generate_series(1,500));
gaussdb=# insert into facts values ('2019-02-25', generate_series(1,600));
gaussdb=# insert into facts values ('2019-02-26', generate_series(1,700));
gaussdb=# insert into facts values ('2019-02-27', generate_series(1,800));

-- Create another table and specify an HLL column:
gaussdb=# create table daily_uniques (
    date      date UNIQUE,
    users     hll
);

-- Group data by date and insert the data into the HLL:
gaussdb=# insert into daily_uniques(date, users)
select date, hll_add_agg(hll_hash_integer(user_id))
from facts
group by 1;

-- Calculate the numbers of users visiting the website every day.
gaussdb=# select date, hll_cardinality(users) from daily_uniques order by date;
 date | hll_cardinality
-----+-----
2019-02-20 |          100
2019-02-21 | 200.217913059312
2019-02-22 | 301.76494508014
2019-02-23 | 400.862858326446
2019-02-24 | 502.626933349694
2019-02-25 | 601.922606454213
2019-02-26 | 696.602316769498
2019-02-27 | 798.111731634412
(8 rows)

-- Calculate the number of users who had visited the website in the week from February 20, 2019 to
February 26, 2019.
gaussdb=# select hll_cardinality(hll_union_agg(users)) from daily_uniques where date >=
'2019-02-20'::date and date <= '2019-02-26'::date;
 hll_cardinality
-----
696.602316769498
(1 row)

-- Calculate the number of users who had visited the website yesterday but have not visited the
website today:
gaussdb=# SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques FROM
daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1 PRECEDING);
-- The default compatibility (MySQL compatibility) results are as follows:
 date | lost_uniques
```

```
-----+-----  
2019-02-20 |      0  
2019-02-21 |      0  
2019-02-22 |      0  
2019-02-23 |      0  
2019-02-24 |      0  
2019-02-25 |      0  
2019-02-26 |      0  
2019-02-27 |      0  
(8 rows)
```

```
-- Drop the table.  
gaussdb=# drop table facts;  
gaussdb=# drop table daily_uniques;
```

- Scenario 3: The data to be inserted does not meet the requirements of the HLL data structure.

When inserting data into a column of the HLL type, ensure that the data meets the requirements of the HLL data structure. If the data does not meet the requirements after being parsed, an error will be reported. In the following example, 'E\1234' to be inserted does not meet the requirements of the HLL data structure after being parsed. As a result, an error is reported.

```
gaussdb=# create table test(id integer, set hll);  
gaussdb=# insert into test values(1, 'E\1234');  
ERROR: not a hll type, size=6 is not enough  
gaussdb=# drop table test;
```

7.3.13 Range Types

A range type is a data type that represents the range of a value of an element type (called the subtype of a range). For example, the range of timestamp may be used to express a time range in which a conference room is reserved. In this case, the data type is `tsrange` (short for timestamp range), and timestamp is the subtype. The subtype must have an overall order so that the element value can be clearly specified within a range, before, or after.

Range types are useful because they can express multiple element values in a single range value and can clearly express concepts such as range overlapping (time and date ranges for time arrangement, price ranges, and instrument ranges).

Built-in Ranges

The following built-in ranges are available:

- `int4range`: integer range.
- `int8range`: bigint range.
- `numrange`: numeric range.
- `tsrange`: range of timestamp without the time zone.
- `tstzrange`: range of timestamp with the time zone
- `daterange`: date range.

In addition, you can define your own range types. For details, see [CREATE TYPE](#).

Example

```
CREATE TABLE reservation (room int, during tsrange);  
INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
```

```
-- Inclusion
SELECT int4range(10, 20) @> 3;
?column?
-----
f
(1 row)
-- Determine whether the two ranges overlap.
SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);
?column?
-----
t
(1 row)
-- Upper bound extraction
SELECT upper(int8range(15, 25));
upper
-----
25
(1 row)
-- Intersection set
SELECT int4range(10, 20) * int4range(15, 25);
?column?
-----
[15,20)
(1 row)
-- Determine whether the range is empty.
SELECT isempty(numrange(1, 5));
isempty
-----
f
(1 row)
DROP TABLE reservation;
```

See the complete list of operators and functions on a range type in [Range Functions and Operators](#).

Including and Excluding Bounds

Each non-empty range has two bounds, a lower bound and an upper bound. All values between the upper and lower bounds are included in the range. An inclusion bound means that the bound value itself is included in the range, while an exclusion bound means that the bound value is not included in the range.

In a textual form of a range, an inclusion lower bound is expressed as "[" and an exclusion lower bound is expressed as "(" . Similarly, one including the upper bound is expressed as "]" and one excluding the upper bound is expressed as ")" (for details, see [Range Input/Output](#)).

The lower_inc and upper_inc functions test the upper and lower bounds of a range value, respectively.

Infinite (Unbounded) Range

When the lower bound of a range is unbounded, it means that all values less than the upper bound are included in the range. Similarly, when the upper bound of a range is unbounded, all values greater than the lower bound are included in the range. When both the upper and lower bounds are unbounded, all values of the element type are considered within the range. The missing bounds are automatically converted to exclusions. You can think of these missing values as positive infinity or negative infinity, but they are special range type values and are considered to be positive and negative infinity values that go beyond any range element type.

Element types with the infinity values can be used as explicit bound values. For example, in the timestamp range, [today,infinity) does not include a special timestamp value infinity.

The lower_inf and upper_inf functions test the infinite upper and lower bounds of a range, respectively.

Range Input/Output

The input of a range value must follow one of the following formats:

```
(lower-bound,upper-bound)
(lower-bound,upper-bound]
[lower-bound,upper-bound)
[lower-bound,upper-bound]
empty
```

The output of a range value must follow one of the following formats:

```
[lower-bound, upper-bound)
Empty
```

Parentheses () or square brackets [] indicate whether the upper and lower bounds are excluded or included, respectively. Note that the last format is empty, which represents an empty range (a range that does not contain values).

The value of *lower-bound* can be a valid input string of the subtype or null, indicating that there is no lower bound. Similarly, *upper-bound* can be a valid input string of the subtype or null, indicating that there is no upper bound.

Each bound value can be referenced using the quotation marks("") character. This is necessary if the bounds value contains parentheses (), square brackets [], commas (,), quotation marks (""), or backslashes (\). Otherwise, those characters will be considered as the part of the range syntax. To put the quotation mark or backslash in a referenced bound value, put a backslash in front of it (and a pair of double quotation marks in its quoted bound value represents one quotation mark character, which is similar to the single quotation mark rule in SQL character strings). In addition, you can avoid referencing and use backslash escapes to protect all data characters, otherwise they will be used as part of the return syntax. Also, if you want to write a bound value that is an empty string, write "", indicating infinite bounds.

Spaces are allowed before and after a range value, but any space between parentheses or square brackets is used as part of the upper or lower bound value (depending on the element type, the space may or may not represent a value).

Examples:

```
-- All values between 3 (included) and 7 (excluded) are included.
gaussdb=# SELECT '[3,7)>::int4range;
int4range
-----
[3,7)
(1 row)
-- All values between 3 (excluded) and 7 (excluded) are included.
gaussdb=# SELECT '(3,7)>::int4range;
int4range
-----
[4,7)
(1 row)
-- Only value 4 is included.
gaussdb=# SELECT '[4,4]>::int4range;
```

```
int4range
-----
[4,5)
(1 row)
-- No value is included (and the value will be normalized to null).
gaussdb=# SELECT '[4,4)':int4range;
int4range
-----
Empty
(1 row)
```

Constructing Range

Each range type has a constructor function with the same name. Using constructor functions is often more convenient than writing a range literal constant because it avoids extra references to bound values. Constructor functions accept two or three parameters. Two parameters form a range in the standard form, where the lower bound is included and the upper bound is excluded, and three parameters form a range according to the bound specified by the third parameter. The third parameter must be one of the following character strings: (), [], or []. For example:

```
-- The complete format is: lower bound, upper bound, and textual parameters indicating the inclusion/
exclusion of bounds.
gaussdb=# SELECT numrange(1.0, 14.0, '[');
numrange
-----
(1.0,14.0]
(1 row)
-- If the third parameter is ignored, it is assumed to be '['.
gaussdb=# SELECT numrange(1.0, 14.0);
numrange
-----
[1.0,14.0)
(1 row)
-- Although '[' is specified here, the value will be converted to the standard format when displayed,
because int8range is a discrete range type.
gaussdb=# SELECT int8range(1, 14, '[');
int8range
-----
[2,15)
(1 row)
-- Using NULL for a bound causes the range to be unbounded on that side.
gaussdb=# SELECT numrange(NULL, 2.2);
numrange
-----
(,2.2)
(1 row)
```

Discrete Range

A range element type has a well-defined "step" such as integer or date. In these types, if there is no valid value between two elements, they can be said to be adjacent. This is in contrast to a continuous range in which other element values can always be identified between two given values. For example, a range above the numeric type is continuous, and the range of timestamp is also continuous. (Although timestamp has limited precision and can be considered as discrete in theory, it can be considered as continuous because the step is not normally considered.)

Another way to consider discrete range types is to have a clear "next" or "previous" value for each element value. With this idea in mind, you can switch

between inclusion and exclusion expressions of a range bound by replacing it with the original given next or previous element value. For example, in an integer range type, [4,8] and (3,9) represent the same set of values, but not for numeric ranges.

A discrete range type should have a regularization function that knows the expected step size of the element type. The regularization function can convert the equivalents of the range type to the same expression, in particular consistent with the inclusion or exclusion bounds. If you do not specify a regularization function, ranges with different formats will always be considered as unequal, even if they actually express the same set of values.

The built-in range types `int4range`, `int8range`, and `daterange` use a regularized form that includes the lower bound and excludes the upper bound, that is, []. However, user-defined range types can use other conventions.

Index

In addition, B-tree indexed can be created on table columns of the range type. For these index types, basically the only useful range operation is equivalence. Using the corresponding < and > operators, there is a B-tree sort order for range value definitions, but that order is fairly arbitrary and is often less useful in the reality. The B-tree support for range types is primarily designed to allow sorting within a query, rather than creating an index.

7.3.14 Object Identifier Types

Object identifiers (OIDs) are used internally by GaussDB as primary keys for various system catalogs. OIDs are not added to user-created tables by the system. The OID type represents an object identifier.

The OID type is currently implemented as an unsigned four-byte integer. Therefore, you are advised not to use a OID column in the created table as a primary key.

Table 7-22 Object identifier types

Name	Reference	Description	Example
OID	N/A	Numeric object identifier	564182
CID	N/A	Command identifier. This is the data type of the system columns cmin and cmax . Command identifiers are 32-bit quantities.	N/A
XID	N/A	A transaction identifier. This is the data type of the system columns xmin and xmax . Transaction identifiers are 64-bit quantities.	N/A

Name	Reference	Description	Example
TID	N/A	Row identifier. This is the data type of the system column ctid . A row ID is a pair (block number, tuple index within block) that identifies the physical location of the row within its table.	N/A
REGCONFIG	pg_ts_config	Text search configuration	english
REGDICTIONARY	pg_ts_dict	Text search dictionary	simple
REGOPER	pg_operator	Operator name	N/A
REGOPERATOR	pg_operator	Operator with argument types	*(integer,integer) or -(NONE,integer)
REGPROC	pg_proc	Function name	sum
REGPROCEDURE	pg_proc	Function with argument types	sum(int4)
REGCLASS	pg_class	Relation name	pg_type
REGTYPE	pg_type	Data type name	integer

The OID type is used for a column in the database system catalog.

Example:

```
gaussdb=# SELECT oid FROM pg_class WHERE relname = 'pg_type';
oid
-----
1247
(1 row)
```

The alias type for OID is **REGCLASS** which allows simplified search for OID values.

Example:

```
gaussdb=# SELECT attrelid,attname,attypid,attstattarget FROM pg_attribute WHERE attrelid =
'pg_type'::REGCLASS;
attrelid | attname | attypid | attstattarget
-----+-----+-----+-----
1247 | xc_node_id | 23 | 0
1247 | tableoid | 26 | 0
1247 | cmax | 29 | 0
1247 | xmax | 28 | 0
1247 | cmin | 29 | 0
1247 | xmin | 28 | 0
1247 | oid | 26 | 0
1247 | ctid | 27 | 0
1247 | typname | 19 | -1
```

1247	typnamespace	26	-1
1247	typowner	26	-1
1247	typplen	21	-1
1247	typbyval	16	-1
1247	typstype	18	-1
1247	typcategory	18	-1
1247	typispreferred	16	-1
1247	typisdefined	16	-1
1247	typdelim	18	-1
1247	typrelid	26	-1
1247	typelem	26	-1
1247	typarray	26	-1
1247	typinput	24	-1
1247	typoutput	24	-1
1247	typreceive	24	-1
1247	typsend	24	-1
1247	typmodin	24	-1
1247	typmodout	24	-1
1247	typanalyze	24	-1
1247	typalign	18	-1
1247	typstorage	18	-1
1247	typnotnull	16	-1
1247	typbasetype	26	-1
1247	typmod	23	-1
1247	typndims	23	-1
1247	typcollation	26	-1
1247	typdefaultbin	194	-1
1247	typdefault	25	-1
1247	typacl	1034	-1
1247	typelemmod	23	-1

(39 rows)

7.3.15 Pseudo-Types

GaussDB type system contains a number of special-purpose entries that are collectively called pseudo-types. A pseudo-type cannot be used as a column data type, but it can be used to declare a function's argument or result type.

Each of the available pseudo-types is useful in situations where a function's behavior does not correspond to simply taking or returning a value of a specific SQL data type. [Table 7-23](#) lists all pseudo-types.

Table 7-23 Pseudo-types

Name	Description
any	Indicates that a function accepts any input data type.
anyelement	Indicates that a function accepts any data type.
anyarray	Indicates that a function accepts any array data type.
anynonarray	Indicates that a function accepts any non-array data type.
anyenum	Indicates that a function accepts any enum data type.
anyrange	Indicates that a function accepts any range data type.
cstring	Indicates that a function accepts or returns a null-terminated C string.
internal	Indicates that a function accepts or returns a server-internal data type.

Name	Description
language_handler	Indicates that a procedural language call handler is declared to return language_handler .
fdw_handler	Indicates that a foreign-data wrapper handler is declared to return fdw_handler .
record	Identifies a function returning an unspecified row type.
trigger	Indicates that a trigger function is declared to return trigger .
void	Indicates that a function returns no value.
opaque	Indicates an obsolete type name that formerly served all the above purposes.

Functions coded in C (whether built in or dynamically loaded) can be declared to accept or return any of these pseudo data types. It is up to the function author to ensure that the function will behave safely when a pseudo-type is used as an argument type.

Functions coded in procedural languages can use pseudo-types only as allowed by their implementation languages. At present the procedural languages all forbid use of a pseudo-type as argument type, and allow only **void** and **record** as a result type. Some also support polymorphic functions using the anyelement, anyarray, anynonarray, anyenum, and anyrange types.

Each location (parameter or return value) declared as the anyelement type is allowed to have any specific actual data type, but they must all be of the same actual type in any given query.

The internal pseudo-type is used to declare functions that are meant only to be called internally by the database system, and not by direct call in an SQL query. If a function has at least one internal-type argument then it cannot be called from SQL. You are advised not to create any function that is declared to return internal unless it has at least one internal argument.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE t1 (a int);

-- Insert two data records.
gaussdb=# INSERT INTO t1 values(1),(2);

-- Create the showall() function.
gaussdb=# CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from t1; $$
LANGUAGE SQL;

-- Call the showall() function.
gaussdb=# SELECT showall();
showall
-----
(2)
(1 row)
```

```
-- Delete the function.
gaussdb=# DROP FUNCTION showall();

-- Delete the table.
gaussdb=# DROP TABLE t1;
```

7.3.16 XML Type

The XML data type can be used to store Extensible Markup Language (XML) data. The internal format of XML is the same as that of the TEXT data type. Its advantage over storing XML data in a TEXT field is that, XML data supports standard XML operation functions based on LIBXML2 and XML standardization check.

The XML data type can store well-formed documents as defined by the XML standard, as well as content fragments, which are defined by referencing broader "DOCUMENT NODE" XQuery and XPath data models. Roughly speaking, this means that there can be more than one top-level element or character node in a content fragment. The expression XMLVALUE IS DOCUMENT can be used to evaluate whether a particular XML value is a complete document or just a document fragment.

The XML parser converts an XML document into an XML DOM object. The document object model (DOM) defines standard methods for accessing and manipulating documents. XML DOM defines standard methods for accessing and manipulating XML documents. XML DOM views XML documents as a tree structure. All elements can be accessed through the DOM tree. You can modify or delete their contents and create new elements. Elements, their text, and their attributes are considered as nodes.

The XML bottom layer uses the same data structure as the TEXT type for storage. The maximum size is 1 GB.

Example:

```
gaussdb=# CREATE TABLE xmltest ( id int, data xml );
gaussdb=# INSERT INTO xmltest VALUES (1, 'one');
gaussdb=# INSERT INTO xmltest VALUES (2, 'two');
gaussdb=# SELECT * FROM xmltest ORDER BY 1;
 id | data
----+-----
 1 | one
 2 | two
(2 rows)
gaussdb=# SELECT xmlconcat(xmlcomment('hello'),
                           xmlelement(NAME qux, 'xml!'),
                           xmlcomment('world'));
          xmlconcat
-----
<!--hello--><qux>xml</qux><!--world-->
(1 row)
gaussdb=# DROP TABLE xmltest;
```

 NOTE

- The XML type does not support the following operations:
 - Logical expressions AND, OR, and NOT
 - Input parameter of a system function that is used as a non-XML operation function
 - Used as a distribution key, partition key, level-2 partition key, primary key, or unique constraint.
 - Implicit conversion related to XML, including the conversion between strings and the XML data type.
 - Array expression, row expression, subquery expression, TABLE OF, and TABLE OF INDEX.
 - Use columns of the XML data format as ordinary indexes, unique indexes, global indexes, local indexes, and partial indexes.
 - Comparison expressions >, <, >=, <=, =, <>, !=, ^=, <=>, BETWEEN AND, IS DISTINCT FROM, and IS NOT DISTINCT FROM
 - Condition expressions DECODE, NULLIF, GREATEST, and LEAST.
 - Used as DISTINCT, GROUP BY, or ORDER BY parameters.
 - Aggregate functions including sum, max, min, avg, list_agg, corr, covar_pop, covar_samp, stddev, stddev_pop, stddev_samp, var_pop, var_samp, variance, bit_and, bit_or, bool_and, bool_or, every, regr_avgx, regr_avgy, regr_count, regr_intercept, regr_r2, regr_slope, regr_sxx, regr_sxy, regr_syy, rank, and spread
 - ODBC-related APIs with binding parameters
- The XML type supports the following operations:
 - Physical backup and restoration
 - Comparison expressions IS NULL and IS NOT NULL
 - Condition expressions CASE and COALESCE
 - Global temporary tables and local temporary tables
 - Forcible type conversion
 - Expression indexes
 - Input XML values that comply with the XML standard
 - gs_dump and gs_restore
 - Parallel query, supporting Astore and Ustore.
 - Input parameters, output parameters, customized variables, and return values of a user-defined function
 - Input parameters, output parameters, customized variables, and return values of a stored procedure, as well as stored procedures that support autonomous transactions.
 - Character processing function quote_literal(string text) (explicitly converted to the character type) and quote_nullable(string text) (explicitly converted to the character type)
 - Aggregate functions count, array_agg, and checksum (explicitly converted to the character type), and string_agg (explicitly converted to the character type)
 - JDBC and ODBC operations on XML data types are supported. The SELECT, UPDATE, INSERT, and DELETE operations can be performed on an XML column. You can enter an XML value using the SQL syntax and use the getSQLXML method of the ResultSet class to obtain the XML value. JDBC-related APIs with binding parameters are supported. For example, you can use the setSQLXML method in the PreparedStatement preprocessing API and the getSQLXML(int columnIndex) method in the ResultSet execution result set API.

In the calling process, use the java.sql.SQLXML API class to construct an XML object, set the specified object type to Oid.XML, and send the type ID and XML value to the server. After obtaining the result returned from the server, call

ResultSet.getString. Then, use the java.sql.SQLXML API class to construct an XML object based on the obtained character string. In this case, the system checks whether the content complies with the XML standard again. Therefore, you can also use ResultSet.getString to directly obtain the XML string object.

7.3.17 XMLType

The XMLType data type is used to store XMLType data. Currently, data is stored in character strings in the internal format. Its advantage over storing XML data in a TEXT field is that, XML data supports standard XML operation functions based on LIBXML2 and XML standardization check.

The XMLType type can store well-formed "documents" that comply with the XML standard.

The XML parser converts an XML document into an XML DOM object. The document object model (DOM) defines standard methods for accessing and manipulating documents. XML DOM defines standard methods for accessing and manipulating XML documents. XML DOM views XML documents as a tree structure. All elements can be accessed through the DOM tree. You can modify or delete their contents and create new elements. Elements, their text, and their attributes are considered as nodes. The maximum size is 1 GB.

Example:

```
gaussdb=# CREATE TABLE xmltypetest(id int, data xmltype);
gaussdb=# INSERT INTO xmltypetest VALUES (1, '<ss/>');
gaussdb=# INSERT INTO xmltypetest VALUES (2, '<xx/>');
gaussdb=# SELECT * FROM xmltypetest ORDER BY 1;
 id | data
----+-----
  1 | <ss/>
  2 | <xx/>
(2 rows)
gaussdb=# DROP TABLE xmltest;
```

 NOTE

- The XMLType type does not support the following operations:
 - Logical expressions AND, OR, and NOT
 - Input parameter of a system function that is used as a non-XMLType operation function
 - Used as a distribution key, partition key, level-2 partition key, primary key, or unique constraint.
 - Implicit conversion related to XMLType, including the conversion between strings and the XMLType data type.
 - Array expression, row expression, and subquery expression.
 - Use columns of the XMLType data format as ordinary indexes, unique indexes, global indexes, local indexes, and partial indexes.
 - Comparison expressions >, <, >=, <=, =, <>, !=, ^=, <=>, BETWEEN AND, IS DISTINCT FROM, and IS NOT DISTINCT FROM
 - Condition expressions DECODE, NULLIF, GREATEST, and LEAST
 - Used as DISTINCT, GROUP BY, or ORDER BY parameters
 - Aggregate functions sum, max, min, avg, list_agg, corr, covar_pop, covar_samp, stddev, stddev_pop, stddev_samp, var_pop, var_samp, variance, bit_and, bit_or, bool_and, bool_or, every, regr_avgx, regr_avgy, regr_count, regr_intercept, regr_r2, regr_slope, regr_sxx, regr_sxy, regr_syy, rank, and spread
 - ODBC-related APIs with binding parameters
- The XMLType type supports the following operations:
 - Physical backup and restoration
 - Comparison expressions IS NULL and IS NOT NULL
 - Condition expressions CASE and COALESCE
 - Global temporary tables and local temporary tables
 - Forcible type conversion
 - Expression indexes
 - Input XMLType values that comply with the XML standard
 - gs_dump and gs_restore
 - Parallel query, supporting Astore and Ustore.
 - Input parameters, output parameters, customized variables, and return values of a user-defined function
 - Input parameters, output parameters, customized variables, and return values of a stored procedure, as well as stored procedures that support autonomous transactions.
 - Character processing function quote_literal(string text) (explicitly converted to the character type) and quote_nullable(string text) (explicitly converted to the character type)
 - Aggregate functions count, array_agg, and checksum (explicitly converted to the character type), and string_agg (explicitly converted to the character type)
 - The SELECT, UPDATE, INSERT, and DELETE operations can be performed on an XMLType column. You can enter an XMLType value using the SQL syntax.
- You can create a schema named **xmltype**. In the schema, you can create functions, but cannot use schema.func() to call functions defined in the schema.

7.3.18 ACLItem

The aclitem data type is used to store object permission information. Its internal implementation is of the int type and supports the *user1=privs/user2* format.

The `aclitem[]` data type is an array consisting of ACL items. The supported format is `{user1 = privs1/user3, user2 = privs2/user3}`.

In the preceding command, `user1`, `user2`, and `user3` indicate the existing users or roles in the database, and `privs` indicates the permissions supported by the database. For details, see [Table 12-44](#).

Example:

```
-- Create a user.
gaussdb=# CREATE USER user1 WITH PASSWORD 'Aa123456789';
gaussdb=# CREATE USER user2 WITH PASSWORD 'Aa123456789';
gaussdb=# CREATE USER omm WITH PASSWORD 'Aa123456789';

-- Create a data table table_acl that contains three columns of the int, aclitem, and aclitem[] types.
gaussdb=# CREATE TABLE table_acl (id int,priv aclitem,privs aclitem[]);
-- Insert a data record whose content is (1,'user1=arw/omm','{omm=d/user2,omm=w/omm}') into the
table_acl table.
gaussdb=# INSERT INTO table_acl VALUES (1,'user1=arw/omm','{omm=d/user2,omm=w/omm}');
-- Insert a data record whose content is (2,'user1=aw/omm','{omm=d/user2}') into the table_acl table.
gaussdb=# INSERT INTO table_acl VALUES (2,'user1=aw/omm','{omm=d/user2}');

gaussdb=# SELECT * FROM table_acl;
id | priv | privs
-----+-----+-----
 1 | user1=arw/omm | {omm=d/user2,omm=w/omm}
 2 | user1=aw/omm | {omm=d/user2}
(2 rows)

-- Delete the table and user.
gaussdb=# DROP USER user1;
gaussdb=# DROP USER user2;
gaussdb=# DROP USER omm;
gaussdb=# DROP TABLE table_acl;
```

7.3.19 Array Types

Array types can be used to store several elements of the same type.

Array Type Definition

Generally, an array data type is named by adding square brackets ([]) to the end of the data type name of an array element.

Example 1: Create a table named **sal_emp**. The table has the following columns: **name** of the text type, indicating employee names; **pay_by_quarter** of the integer element type, indicating an array of quarterly salaries; **phone_numbers** of the `varchar(11)` element type, indicating an array of mobile numbers.

```
gaussdb=# CREATE TABLE sal_emp (
name      text,
pay_by_quarter integer[], phone_numbers varchar(11)[]
);
gaussdb=# DROP TABLE sal_emp;
```

Example 2: Define an array type in other ways. For details about the definition method and behavior, see the comments in the example.

```
gaussdb=# CREATE TABLE sal_emp (
name      text,
pay_by_quarter1 integer[], -- Two-dimensional array of the int type.
pay_by_quarter2 integer[3], -- One-dimensional array of the int type. The size is 3.
pay_by_quarter3 integer[3][3], -- Two-dimensional array of the int type. The size of each dimension is 3.
pay_by_quarter4 integer ARRAY, -- One-dimensional array of the int type.
```

```
pay_by_quarter5 integer ARRAY[3] -- One-dimensional array of the int type. The size is 3.
gaussdb=# DROP TABLE sal_emp;
```

⚠ CAUTION

- The definition of the number of dimensions of an array does not take effect (which does not affect the runtime behavior). You are advised to use the method described in example 1 to define the array type and do not use multidimensional array data.
- The definition of the array size does not take effect (which does not affect the runtime behavior). You are advised to use the method described in example 1 to define the array type.
- The maximum number of dimensions of an array is 6.
- The restrictions on the number of array elements are as follows:
 1. The maximum number of elements is 134217727.
 2. The maximum storage space of all elements cannot exceed 1 GB minus 1 byte, that is, 1073741823 bytes.

Array Constructor

An array constructor is an expression that can build array values. A simple array constructor consists of the keyword ARRAY and an expression list of array element values which are separated by commas and enclosed (,) by square brackets ([]). For example:

```
gaussdb=# SELECT ARRAY[1, 2, 3 + 4];
array
-----
{1,2,7}
(1 row)
```

By default, the element type of an array is the common type of member expressions and is determined by the same rules as the UNION or CASE structure ([UNION, CASE, and Related Constructs](#)). You can construct an array to the desired data type through explicit type conversion. The following is an example:

```
gaussdb=# SELECT ARRAY[1, 2, 3]::varchar[];
array
-----
{1,2,3}
(1 row)

gaussdb=# SELECT ARRAY['a', 'b', 'c']::varchar[];
array
-----
{a,b,c}
(1 row)
```

In addition to the preset basic types, the table type can also be defined as array types. The following is an example:

```
gaussdb=# CREATE TYPE rec IS (c1 int, c2 int);
gaussdb=# SELECT ARRAY[(1, 1), (2, 2)]::rec[];
array
-----
{"(1,1)","(2,2)"}
(1 row)
```

```
gaussdb=# CREATE TABLE tab (c1 int, c2 int);
gaussdb=# SELECT ARRAY[(1, 1), (2, 2)::tab[];
array
```

```
-----
{"(1,1)","(2,2)"}
(1 row)
```

```
gaussdb=# DROP TYPE rec;
gaussdb=# DROP TABLE tab;
```

An array must have a type. Therefore, when constructing an empty array, you must construct it as the required type. For example:

```
gaussdb=# SELECT ARRAY[]::int[];
array
```

```
-----
{}
(1 row)
```

You can also construct an array from the result of a subquery. In this case, the array constructor consists of the keyword ARRAY and a subquery enclosed in parentheses. The subquery must return only one separate column. The generated one-dimensional array generates an element for each row of the result in the subquery. The element type matches the output column of the subquery. For example:

```
gaussdb=# SELECT ARRAY(SELECT generate_series(1, 6));
array
```

```
-----
{1,2,3,4,5,6}
(1 row)
```

Multidimensional array values can be made by nesting array constructors. The ARRAY keyword in the inner constructor can be omitted. For example, the following two examples show the same result:

```
gaussdb=# SELECT ARRAY[ARRAY[1,2], ARRAY[3,4]];
array
```

```
-----
{{1,2},{3,4}}
(1 row)
```

```
gaussdb=# SELECT ARRAY[[1,2], [3,4]];
array
```

```
-----
{{1,2},{3,4}}
(1 row)
```

NOTE

- Inner constructors at the same layer must generate subarrays of the same dimension.
- Any type conversion applied to the outer ARRAY constructor is automatically applied to all inner constructors.

String Inputs of the Array Type

To write an array value as a literal constant (constant input), enclose the element values with braces and separate them with commas. The general format of an array constant is as follows:

```
'{ val1 delim val2 delim ... }'
```

In the preceding format, **delim** indicates the delimiter of the element type and is recorded in the **typpdelim** column in the pg_type catalog of the type. Each **val** can be a constant of the array element type or a subarray. For example:

```
gaussdb=# SELECT '{1, 2, 3}':int[] AS RESULT;
result
-----
{1,2,3}
(1 row)

gaussdb=# SELECT '{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}':int[] AS RESULT;
result
-----
{{1,2,3},{4,5,6},{7,8,9}}
(1 row)
```

Double quotation marks can be used around any element value, and double quotation marks must be used when the element value contains special characters such as commas or braces. The following is an example:

```
gaussdb=# SELECT '{" ", "NULL", null, "\\ ", "{", "}", ", "}'::varchar[] AS RESULT;
result
-----
{" ", "NULL", NULL, "\\ ", "{", "}", ", " }
(1 row)
```

-- This example indicates that there is an array of the varchar type and there are seven varchar elements in total. The elements are as follows:

1. A character string containing a space.
2. A character string containing "NULL".
3. A character string containing NULL.
4. A character string containing a backslash (\).
5. A character string containing a left brace {.
6. A character string containing a right brace }.
7. A character string containing a comma (,).

NOTE

- For array string constant inputs, if the array element value is an empty string, contains braces, delimiters, double quotation marks, backslashes, or spaces, or matches the keyword NULL, then element inputs are enclosed in double quotation marks. An additional backslash is input if the element value contains double quotation marks or backslashes.
- The keyword NULL is case-insensitive.
- The input spaces not enclosed in double quotation marks are automatically skipped.
- You are advised to use the array constructor instead of character constants to construct array data.

String Outputs of the Array Type

The output of an array value consists of the output of the array element type plus some modifiers indicating the array structure. These modifiers consist of braces ({}), around array values plus delimiters between adjacent items. In a multidimensional array, each dimension has its own level of braces and contains delimiters between adjacent braces at the same level.

The data of the array type contains special characters (in the following description). The following is an example of character string output:

```
gaussdb=# SELECT ARRAY['{', '}', 'hello, world', '', '\', ' ', NULL] AS RESULT;
array
-----
{"{", "}", "hello, world", "\\ ", "\\", " ", NULL}
(1 row)
```

 NOTE

For array string constant outputs, if the array element value is an empty string or contains braces, delimiters, double quotation marks, backslashes, or spaces, or the element is NULL, then element outputs are enclosed in double quotation marks. An additional backslash is output if the element value contains double quotation marks or backslashes. This parameter corresponds to the string constant inputs.

Use of Array Types

The following is an example of using the array type:

```
-- Create a table with array columns and insert some data.
gaussdb=# CREATE TABLE orders (
  name varchar,
  items varchar[]
);
gaussdb=# INSERT INTO orders VALUES('a', ARRAY['Apple', 'Orange', 'Pear']);
gaussdb=# INSERT INTO orders VALUES('b', ARRAY['Mineral water', 'Coke', 'Sprite']);
gaussdb=# INSERT INTO orders VALUES('c', ARRAY['Mouse', 'Keyboard', 'Earphone']);
gaussdb=# INSERT INTO orders VALUES('d', '{Cabbage, Potato, Eggplant}');

-- Query data.
gaussdb=# SELECT * FROM orders ORDER BY name;
 name |      items
-----+-----
 a    | {Apple,Orange,Pear}
 b    | {Mineral water,Coke,Sprite}
 c    | {Mouse,Keyboard,Earphone}
 d    | {Cabbage,Potato,Eggplant}
(4 rows)

-- Access array elements.
gaussdb=# SELECT items[1] FROM orders ORDER BY name;
 items
-----
 Apple
Mineral water
 Mouse
 Cabbage
(4 rows)

-- If the accessed element exceeds the range or the accessed index is NULL, NULL is returned.
gaussdb=# SELECT items[4] FROM orders ORDER BY name;
 items
-----

(4 rows)

gaussdb=# SELECT items[null] FROM orders ORDER BY name;
 items
-----

(4 rows)

-- Access a subarray.
gaussdb=# SELECT items[1:2] FROM orders ORDER BY name;
 items
-----
 {Apple,Orange}
 {Mineral water,Coke}
 {Mouse,Keyboard}
```

```

{Cabbage,Potato}
(4 rows)

-- Updates the entire array.
gaussdb=# UPDATE orders SET items = ARRAY['Banana', 'Watermelon', 'Strawberry'] WHERE name = 'a';
gaussdb=# SELECT items FROM orders WHERE name = 'a';
        items
-----
{Banana,Watermelon,Strawberry}
(1 row)

-- Updates the elements of an array.
gaussdb=# UPDATE orders SET items[1] = 'Mango' WHERE name = 'a';
gaussdb=# SELECT items FROM orders WHERE name = 'a';
        items
-----
{Mango,Watermelon,Strawberry}
(1 row)

-- Updates the element fragment of an array.
gaussdb=# UPDATE ORDERS SET items[1:2] = ARRAY['Computer', 'Mobile phone'] WHERE name = 'c';
gaussdb=# SELECT items FROM ORDERS WHERE name = 'c';
        items
-----
{Computer,Mobile phone,Earphone}
(1 row)

-- Add an array element. All unassigned elements between the last element of the original array and the
new element are set to NULL.
gaussdb=# UPDATE orders SET items[4] = 'Display' WHERE name = 'c';
gaussdb=# SELECT items FROM orders WHERE name = 'c';
        items
-----
{Computer,Mobile phone,Earphone,Display}
(1 row)

gaussdb=# UPDATE orders SET items[6] = 'Display 2' WHERE name = 'c';
gaussdb=# SELECT items FROM orders WHERE name = 'c';
        items
-----
{Computer,Mobile phone,Earphone,Display,NULL,Display 2}
(1 row)

gaussdb=# DROP TABLE orders;

```

7.4 Constants and Macros

[Table 7-24](#) lists the constants and macros that can be used in GaussDB.

Table 7-24 Constants and macros

Parameter	Description	Example
CURRENT_CATALOG	Specifies the current database.	testdb=# SELECT CURRENT_CATALOG; current_database ----- testdb (1 row)
CURRENT_ROLE	Specifies the current user.	gaussdb=# SELECT CURRENT_ROLE; current_user ----- omm (1 row)

Parameter	Description	Example
CURRENT_SCHEMA	Specifies the current database mode.	gaussdb=# SELECT CURRENT_SCHEMA; current_schema ----- public (1 row)
CURRENT_USER	Specifies the current user.	gaussdb=# SELECT CURRENT_USER; current_user ----- omm (1 row)
LOCALTIMESTAMP	Specifies the current session time (without time zone).	gaussdb=# SELECT LOCALTIMESTAMP; timestamp ----- 2015-10-10 15:37:30.968538 (1 row)
NULL	This parameter is left blank.	N/A
SESSION_USER	Specifies the current system user.	gaussdb=# SELECT SESSION_USER; session_user ----- omm (1 row)
SYSDATE	Specifies the current system date.	gaussdb=# SELECT SYSDATE; sysdate ----- 2015-10-10 15:48:53 (1 row)
USER	Specifies the current user, also called CURRENT_USER .	gaussdb=# SELECT USER; current_user ----- omm (1 row)

7.5 Functions and Operators

Operators can be used to process one or more operands and can be placed before, after, or between operands. Results are returned after the processing.

Functions encapsulate service logic to implement specific functions. A function may or may not have parameters. After a function is executed, the result is returned.

Users can modify system functions. However, after the modification, the meaning of the functions may change, which results in disorder in system control. Therefore, users are not allowed to manually modify system functions.

NOTE

When the GUC parameter **behavior_compat_options** contains the **'enable_funcname_with_argname'** option, the projection alias displays the complete function.

7.5.1 Logical Operators

The usual logical operators include AND, OR, and NOT. SQL uses a three-valued logical system with true, false, and null, which represents "unknown". Their priorities are NOT > AND > OR.

Table 7-25 lists the calculation rules, where a and b represent logical expressions.

Table 7-25 Operation rules

a	b	a AND b Result	a OR b Result	NOT a Result
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	NULL	NULL	NULL	NULL

NOTE

- The operators AND and OR are commutative, that is, you can switch the left and right operand without affecting the result.
- Operations on XML data are not supported.

7.5.2 Comparison Operators

Comparison operators are available for the most data types and return Boolean values.

All comparison operators are binary operators. Only data types that are the same or can be implicitly converted can be compared using comparison operators.

Table 7-26 describes comparison operators provided by GaussDB.

Table 7-26 Comparison operations

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to

Operator	Description
<>, !=, or ^=	Not equal to

- Comparison operators are available for all relevant data types. All comparison operators are binary operators that returned values of Boolean type.
- The calculation priority of the inequality sign is higher than that of the equality sign. If the entered data type is different and cannot be implicitly converted, the comparison fails. For example, an expression such as **1 < 2 < 3** is invalid because the less-than sign (<) cannot be used to compare Boolean values and 3.
- Besides, each comparison operator has a corresponding function in the pg_proc system catalog. If the value of the **proleakproof** attribute of the corresponding function is **f**, the function is not used to prevent data leakage. If a user only has the permission for a system view, but does not have the permission for the corresponding table, the query plan may not be optimal when the user searches the system view.
- This operator does not support data of the XML type.

7.5.3 Character Processing Functions and Operators

String functions and operators provided by GaussDB are for concatenating strings with each other, concatenating strings with non-strings, and matching the patterns of strings. Note: Except length-related functions, other functions and operators of string processing functions do not support parameters of CLOB whose size is greater than 1 GB.

- `bit_length(string)`
Description: Specifies the number of bits occupied by a string.
Return type: int
Example:

```
gaussdb=# SELECT bit_length('world');
 bit_length
-----
         40
(1 row)
```
- `btrim(string text [, characters text])`
Description: Removes the longest string consisting only of characters in **characters** (a space by default) from the start and end of **string**.
Return type: text
Example:

```
gaussdb=# SELECT btrim('string', 'ing');
 btrim
-----
 sr
(1 row)
```
- `char_length(string)` or `character_length(string)`
Description: Specifies the number of characters in a string.
Return type: int

Example:

```
gaussdb=# SELECT char_length('hello');
 char_length
-----
          5
(1 row)
```

- `dump(expr[, return_fmt [, start_position [, length]]])`

Description: Returns the data type code, byte length, and internal representation of the input expression. **return_fmt** specifies the number system of the internal representation, **start_position** specifies the byte from which the internal representation starts, and **length** indicates the length of the data to be read.

Return type: text

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- `instr(text,text,int,int)`

Description: **instr(string1,string2,int1,int2)** returns the text from **int1** to **int2** in **string1**. The first **int** indicates the start position for matching, and the second **int** indicates the number of matching times.

Return type: int

Example:

```
gaussdb=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
 instr
-----
          6
(1 row)
```

- `instrb(text,text,int,int)`

Description: **instrb(string1,string2,int1,int2)** returns the position matching *string2* in *string1* for the *int2*th time from the position specified by *int1*. *int1* indicates the start position for matching, and *int2* indicates the number of matching times. Different from the `instr` function, `instrb` is calculated in bytes and is not affected by the character set in use.

Return type: int

Example:

```
gaussdb=# SELECT instrb( 'abcdabcdabcd', 'bcd', 2, 2 );
 instrb
-----
          6
(1 row)
```

 **NOTE**

- This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s1** in an ORA-compatible database.
 - If the values of *int1* and *int2* are decimals, the values are truncated instead of being rounded off.
- `lengthb(text/bpchar)`

Description: Obtains the number of bytes of a specified string.

Return type: int

Example:

```
gaussdb=# SELECT lengthb('hello');
lengthb
-----
      5
(1 row)
```

- left(str text, n int)

Description: Returns the first *n* characters in a string. When *n* is negative, all but the last **|n|** characters are returned.

Return type: text

Example:

```
gaussdb=# SELECT left('abcde', 2);
left
-----
ab
(1 row)
```

- length(string bytea, encoding name)

Description: Specifies the number of characters in **string** in the given **encoding**. The **string** must be valid in this encoding.

Return type: int

Example:

```
gaussdb=# SELECT length('jose', 'UTF8');
length
-----
      4
(1 row)
```

 **NOTE**

If the length of the bytea type is queried and UTF8 encoding is specified, the maximum length can only be **536870888**.

- lpad(string text, length int [, fill text])

Description: Fills up **string** to **length** by appending the characters **fill** (a space by default). If **string** is already longer than **length**, then it is truncated.

Return type: text

 **NOTE**

In the scenario where this function is in an ORA-compatible database, the value of **a_format_version** is **10c**, and that of **a_format_dev_version** is **s1**:

- The **length** parameter indicates the display length of a character string. The display length of a single character is processed based on ORA compatibility requirements. During the execution of the lpad function, if the remaining length is 1 and the next character is of the full-width type (2 bytes), a space character is added to the left of the string.
 - If the value of *length* is a decimal, the value is truncated instead of being rounded off.
- notlike(x bytea name text, y bytea text)

Description: Compares x and y to check whether they are inconsistent.

Return type: Boolean

Example:

```
gaussdb=# SELECT notlike(1,2);
notlike
-----
t
```

```
(1 row)
gaussdb=# SELECT notlike(1,1);
notlike
-----
      f
(1 row)
```

- `octet_length(string)`

Description: Specifies the number of bytes in a string.

Return type: int

Example:

```
gaussdb=# SELECT octet_length('jose');
octet_length
-----
          4
(1 row)
```

- `overlay(string placing string FROM int [for int])`

Description: Replaces substrings. **FROM int** indicates the start position of the replacement in the first string. **for int** indicates the number of characters replaced in the first string.

Return type: text

Example:

```
gaussdb=# SELECT overlay('hello' placing 'world' from 2 for 3 );
overlay
-----
hworldo
(1 row)
```

- `position(substring in string)`

Description: Specifies the position of a substring. Parameters are case-sensitive.

Return type: int. If the character string does not exist, **0** is returned.

Example:

```
gaussdb=# SELECT position('ing' in 'string');
position
-----
        4
(1 row)
```

- `pg_client_encoding()`

Description: Specifies the current client encoding name.

Return type: name

Example:

```
gaussdb=# SELECT pg_client_encoding();
pg_client_encoding
-----
UTF8
(1 row)
```

- `quote_ident(string text)`

Description: Returns the given string suitably quoted to be used as an identifier in an SQL statement string (quotation marks are used as required). Quotation marks are added only if necessary (that is, if the string contains non-identifier characters or would be case-folded). Embedded quotation marks are properly doubled.

Return type: text

Example:

```
gaussdb=# SELECT quote_ident('hello world');
quote_ident
-----
"hello world"
(1 row)
```

- `quote_literal(string text)`

Description: Returns the given string suitably quoted to be used as text in an SQL statement string (quotation marks are used as required). It supports XML data that is explicitly converted to the character type.

Return type: text

Example:

```
gaussdb=# SELECT quote_literal('hello');
quote_literal
-----
'hello'
(1 row)
```

If a command similar to the following exists, the text will be escaped:

```
gaussdb=# SELECT quote_literal(E'O'hello');
quote_literal
-----
'O"hello'
(1 row)
```

If a command similar to the following exists, the backslash will be properly doubled:

```
gaussdb=# SELECT quote_literal('O\hello');
quote_literal
-----
E'O\\hello'
(1 row)
```

If the parameter is null, **NULL** is returned. If the parameter may be null, you are advised to use **quote_nullable**.

```
gaussdb=# SELECT quote_literal(NULL);
quote_literal
-----
(1 row)
```

- `quote_literal(value anyelement)`

Description: Converts the given value to text and then quotes it as a literal.

Return type: text

Example:

```
gaussdb=# SELECT quote_literal(42.5);
quote_literal
-----
'42.5'
(1 row)
```

If a command similar to the following exists, the given value will be escaped:

```
gaussdb=# SELECT quote_literal(E'O'42.5');
quote_literal
-----
'O"42.5'
(1 row)
```

If a command similar to the following exists, the backslash will be properly doubled:

```
gaussdb=# SELECT quote_literal('O\42.5');
quote_literal
```

```
-----
E'O\42.5'
(1 row)
```

- `quote_nullable(string text)`

Description: Returns the given string suitably quoted to be used as a string in an SQL statement string (quotation marks are used as required).

It supports XML data that is explicitly converted to the character type.

Return type: text

Example:

```
gaussdb=# SELECT quote_nullable('hello');
quote_nullable
```

```
-----
'hello'
(1 row)
```

If a command similar to the following exists, the text will be escaped:

```
gaussdb=# SELECT quote_nullable(E'O\hello');
quote_nullable
```

```
-----
'O"hello'
(1 row)
```

If a command similar to the following exists, the backslash will be properly doubled:

```
gaussdb=# SELECT quote_nullable('O\hello');
quote_nullable
```

```
-----
E'O\\hello'
(1 row)
```

If the parameter is null, **NULL** is returned.

```
gaussdb=# SELECT quote_nullable(NULL);
quote_nullable
```

```
-----
NULL
(1 row)
```

- `quote_nullable(value anyelement)`

Description: Converts the given value to text and then quotes it as a literal.

Return type: text

Example:

```
gaussdb=# SELECT quote_nullable(42.5);
quote_nullable
```

```
-----
'42.5'
(1 row)
```

If a command similar to the following exists, the given value will be escaped:

```
gaussdb=# SELECT quote_nullable(E'O\42.5');
quote_nullable
```

```
-----
'O"42.5'
(1 row)
```

If a command similar to the following exists, the backslash will be properly doubled:

```
gaussdb=# SELECT quote_nullable('O\42.5');
quote_nullable
```

```
-----
E'O\42.5'
(1 row)
```

If the parameter is null, **NULL** is returned.

```
gaussdb=# SELECT quote_nullable(NULL);
quote_nullable
-----
NULL
(1 row)
```

- `substring_inner(string [from int] [for int])`

Description: Extracts a substring. **from int** indicates the start position of the truncation. **for int** indicates the number of bytes truncated.

Return type: text

Example:

```
gaussdb=# SELECT substring_inner('adcde', 2,3);
substring_inner
-----
dcd
(1 row)
```

- `substring(string [from int] [for int])`

Description: Extracts a substring. **from int** indicates the start position of the truncation. **for int** indicates the number of bytes truncated.

Return type: text

Example:

```
gaussdb=# SELECT substring('Thomas' from 2 for 3);
substring
-----
hom
(1 row)
```

- `substring(string from pattern)`

Description: Extracts substrings matching the POSIX-style regular expression. It returns the text that matches the pattern. If no match record is found, a null value is returned.

Return type: text

Example:

```
gaussdb=# SELECT substring('Thomas' from '...$');
substring
-----
mas
(1 row)
gaussdb=# SELECT substring('foobar' from 'o(.)b');
result
-----
o
(1 row)
gaussdb=# SELECT substring('foobar' from '(o(.)b)');
result
-----
oob
(1 row)
```

NOTE

If the POSIX-style regular expression contains any parentheses, the portion of the text that matched the first parenthesized sub-expression (the one whose left parenthesis comes first) is returned. You can put parentheses around the whole expression if you want to use parentheses within it without triggering this exception.

- `substring(string from pattern for escape)`

Description: Extracts substrings matching the SQL regular expression. The declared schema must match the entire data string; otherwise, the function

fails and returns a null value. To indicate the part of the schema that should be returned on success, the schema must contain two occurrences of the escape character followed by a double quotation mark ("). The text matching the portion of the pattern between these marks is returned.

Return type: text

Example:

```
gaussdb=# SELECT substring('Thomas' from '%#"o_a#"_' for '#');
substring
-----
oma
(1 row)
```

- `rawcat(raw,raw)`

Description: Indicates the string concatenation function.

Return type: raw

Example:

```
gaussdb=# SELECT rawcat('ab','cd');
rawcat
-----
ABCD
(1 row)
```

- `regexp_like(text,text,text)`

Description: Indicates the mode matching function of a regular expression.

Return type: bool

Example:

```
gaussdb=# SELECT regexp_like('str','[ac]');
regexp_like
-----
f
(1 row)
```

- `regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]])`

Description: Extracts substrings from a regular expression. Its function is similar to **substr**. When a regular expression contains multiple parallel brackets, it also needs to be processed.

Parameters:

- **string**: source character string used for matching.
- **pattern**: regular expression pattern string used for matching.
- **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
- **occurrence**: sequence number of the matched substring to be extracted. This parameter is optional. The default value is **1**.
- **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. The options supported by **flags** and description are shown in [Table 7-27](#).

Table 7-27 Options supported by flags

Option	Description
'b'	Indicates the BRE matching without extension.

Option	Description
'c'	Indicates the case-sensitive matching.
'e'	Indicates the ERE matching with extension.
'i'	Indicates the case-insensitive matching.
'm'	Indicates the multi-line matching. If flags contains 'm', use the multi-line matching. Otherwise, use the single-line matching.
'n'	<p>The meanings of the 'n' option are related to the GUC parameter behavior_compat_options and the compatibility mode of the current database.</p> <ul style="list-style-type: none"> If the SQL compatibility mode of the database is ORA or MySQL and the value of the GUC parameter behavior_compat_options contains aformat_regexp_match, the 'n' option indicates that "." matches the linefeed '\n'. If 'n' is not specified, "." does not match the linefeed. In other cases, the 'n' option has the same meaning as the 'm' option.
'p'	Indicates partial linefeed-sensitive matching, which is similar to the linefeed-sensitive matching ('m' or 'n') and affects "." and square bracket expression, but does not affect ^ and \$.
'q'	Indicates common character matching.
's'	Indicates the single-line matching. The meaning is opposite to those of 'm' and 'n'.
't'	Indicates the compact matching. The whitespace characters match themselves.
'w'	Indicates the reverse partial linefeed-sensitive matching. The meaning is opposite to that of 'p'.
'x'	Indicates the loose matching. The whitespace characters are ignored.

Return type: text

Example:

```
gaussdb=# SELECT regexp_substr('str','[ac]');
regexp_substr
-----
(1 row)

gaussdb=# SELECT regexp_substr('foobarbaz', 'b(..)', 3, 2) AS RESULT;
result
-----
baz
(1 row)
```

- `regexp_count(string text, pattern text [, position int [, flags text]])`
Description: obtains the number of substrings used for matching.
Parameters:
 - **string**: source character string used for matching.
 - **pattern**: regular expression pattern string used for matching.
 - **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
 - **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. The options supported by **flags** and description are shown in [Table 7-27](#).

 **NOTE**

When the function is in an ORA-compatible database, the value of **a_format_version** is **10c**, and the value of **a_format_dev_version** is **s1**, the **pattern** parameter ending with a backslash (\) is valid.

Return type: int

Example:

```
gaussdb=# SELECT regexp_count('foobarbaz','b(..)', 5) AS RESULT;
result
-----
1
(1 row)
```

- `regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])`
Description: obtains the position (starting from 1) of the substring that meets the matching condition. If no substring is matched, **0** is returned.
Parameters:
 - **string**: source character string used for matching.
 - **pattern**: regular expression pattern string used for matching.
 - **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
 - **occurrence**: sequence number of the matched substring to be replaced. This parameter is optional. The default value is **1**.
 - **return_opt**: specifies whether to return the position of the first or last character of the matched substring. This parameter is optional. If the value is **0**, the position of the first character (starting from 1) of the matched substring is returned. If the value is greater than 0, the position of the next character of the end character of the matched substring is returned. The default value is **0**.
 - **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. The options supported by **flags** and description are shown in [Table 7-27](#).

Return type: int

Example:

```
gaussdb=# SELECT regexp_instr('foobarbaz','b(..)', 1, 1, 0) AS RESULT;
result
-----
4
```

```
(1 row)
gaussdb=# SELECT regexp_instr('foobarbaz','b(..)', 1, 2, 0) AS RESULT;
result
-----
7
(1 row)
```

- `regexp_matches(string text, pattern text [, flags text])`

Description: Returns all captured substrings resulting from matching a POSIX-style regular expression against **string**. If the pattern does not match, the function returns no rows. If the pattern contains no parenthesized sub-expressions, then each row returned is a single-element text array containing the substring matching the whole pattern. If the pattern contains parenthesized sub-expressions, the function returns a text array whose *n*th element is the substring matching the *n*th parenthesized sub-expression of the pattern.

The optional **flags** argument contains zero or multiple single-letter flags that change the function behavior. **i** indicates that the matching is not related to uppercase and lowercase. **g** indicates that each matched substring is replaced, instead of replacing only the first one.

NOTICE

If the last parameter is provided but the parameter value is an empty string (") and the SQL compatibility mode of the database is set to ORA, the returned result is an empty set. This is because the ORA compatible mode treats the empty string (") as **NULL**. To resolve this problem, you can:

- Change the database SQL compatibility mode to TD.
- Do not provide the last parameter or do not set the last parameter to an empty string.

Return type: SETOF text[]

Example:

```
gaussdb=# SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
regexp_matches
-----
{bar,beque}
(1 row)
gaussdb=# SELECT regexp_matches('foobarbequebaz', 'barbeque');
regexp_matches
-----
{barbeque}
(1 row)
gaussdb=# SELECT regexp_matches('foobarbequebazilbarfonk', '(b[^b]+)(b[^b]+)', 'g');
regexp_matches
-----
{bar,beque}
{bazil,barf}
(2 rows)
```

- `regexp_split_to_array(string text, pattern text [, flags text])`

Description: Splits **string** using a POSIX-style regular expression as the delimiter. The **regexp_split_to_array** function behaves the same as **regexp_split_to_table**, except that **regexp_split_to_array** returns its result as an array of text.

Return type: text[]

Example:

```
gaussdb=# SELECT regexp_split_to_array('hello world', E'\\s+');
regexp_split_to_array
-----
{hello,world}
(1 row)
```

- `regexp_split_to_table(string text, pattern text [, flags text])`

Description: Splits **string** using a POSIX-style regular expression as the delimiter. If there is no match to the pattern, the function returns the string. If there is at least one match, for each match it returns the text from the end of the last match (or the beginning of the string) to the beginning of the match. When there are no more matches, it returns the text from the end of the last match to the end of the string.

The **flags** parameter is a text string containing zero or more single-letter flags that change the function's behavior. **i** indicates case-independent matching. If no parameter is contained, each matched substring, not only the first substring, is replaced by default.

Return type: SETOF text

Example:

```
gaussdb=# SELECT regexp_split_to_table('hello world', E'\\s+');
regexp_split_to_table
-----
hello
world
(2 rows)
```

- `repeat(string text, number int)`

Description: Repeats **string** the specified number of times.

Return type: text

Example:

```
gaussdb=# SELECT repeat('Pg', 4);
repeat
-----
PgPgPgPg
(1 row)
```

NOTE

The maximum size of memory allocated at a time cannot exceed 1 GB due to the memory allocation mechanism of the database. Therefore, the maximum value of **number** cannot exceed $(1 \text{ GB} - x) / \text{lengthb}(\text{string}) - 1$. **x** indicates the length of the header information, which is usually greater than 4 bytes. The value varies among different scenarios.

- `replace(string text, from text, to text)`

Description: Replaces all occurrences in **string** of substring **from** with substring **to**.

Return type: text

Example:

```
gaussdb=# SELECT replace('abcdefabcdef', 'cd', 'XXX');
replace
-----
abXXXefabXXXef
(1 row)
```

- `replace(string, substring)`

Description: Deletes all substrings in a string.

String type: text

Substring type: text

Return type: text

Example:

```
gaussdb=# SELECT replace('abcdefabcdef', 'cd');
replace
-----
abefabef
(1 row)
```

- `reverse(str)`

Description: Returns a reversed string (by character).

Return type: text

Example:

```
gaussdb=# SELECT reverse('abcde');
reverse
-----
edcba
(1 row)
```

- `right(str text, n int)`

Description: Returns the last n characters in a string. When n is negative, all but the first $|n|$ characters are returned.

Return type: text

Example:

```
gaussdb=# SELECT right('abcde', 2);
right
-----
de
(1 row)

gaussdb=# SELECT right('abcde', -2);
right
-----
cde
(1 row)
```

- `rpadd(string text, length int [, fill text])`

Description: Fills up **string** to **length** by appending the characters **fill** (a space by default). If **string** is already longer than **length**, then it is truncated.

Return type: text

NOTE

In the scenario where this function is in an ORA-compatible database, the value of **a_format_version** is **10c**, and that of **a_format_dev_version** is **s1**:

- The **length** parameter indicates the display length of a character string. The display length of a single character is processed based on ORA compatibility requirements. During the execution of the `rpadd` function, if the remaining length is 1 and the next character is of the full-width type (2 bytes), a space character is added to the right of the string.
 - If the value of *length* is a decimal, the value is truncated instead of being rounded off.
- `substrb(text,int,int)`

Description: Extracts a substring. The first **int** indicates the start position of the subtraction. The second **int** indicates the number of characters subtracted.

Return type: text

Example:

```
gaussdb=# SELECT substrb('string',2,3);
substrb
-----
tri
(1 row)
```

- substrb(text,int)

Description: Extracts a substring. **int** indicates the start position of the extraction.

Return type: text

Example:

```
gaussdb=# SELECT substrb('string',2);
substrb
-----
tring
(1 row)
```

- substr(bytea,from,count)

Description: Extracts a substring from **bytea**. **from** specifies the position where the extraction starts. **count** specifies the length of the extracted substring.

Return type: text

Example:

```
gaussdb=# SELECT substr('string',2,3);
substr
-----
tri
(1 row)
```

- string || string

Description: Concatenates strings.

Return type: text

Example:

```
gaussdb=# SELECT 'MPP' || 'DB' AS RESULT;
result
-----
MPPDB
(1 row)
```

- string || non-string or non-string || string

Description: Concatenates strings and non-strings.

Return type: text

Example:

```
gaussdb=# SELECT 'Value: ' || 42 AS RESULT;
result
-----
Value: 42
(1 row)
```

- split_part(string text, delimiter text, field int)

Description: Splits **string** on **delimiter** and returns the **fieldth** column (counting from text of the first appeared delimiter).

Return type: text

Example:

```
gaussdb=# SELECT split_part('abc~@~def~@~ghi', '~@~', 2);
split_part
-----
def
(1 row)
```

- `strpos(string, substring)`

Description: Specifies the position of a substring. It is the same as **position(substring in string)**. However, the parameter sequences of them are reversed.

Return type: int

Example:

```
gaussdb=# SELECT strpos('source', 'rc');
strpos
-----
4
(1 row)
```

- `to_hex(number int or bigint)`

Description: Converts a number to a hexadecimal expression.

Return type: text

Example:

```
gaussdb=# SELECT to_hex(2147483647);
to_hex
-----
7fffffff
(1 row)
```

- `translate(string text, from text, to text)`

Description: Any character in **string** that matches a character in **from** is replaced by the corresponding character in **to**. If **from** is longer than **to**, extra characters occurred in **from** are removed.

Return type: text

Example:

```
gaussdb=# SELECT translate('12345', '143', 'ax');
translate
-----
a2x5
(1 row)
```

- `length(string)`

Description: Obtains the number of characters in a string.

Return type: integer

Example:

```
gaussdb=# SELECT length('abcd');
length
-----
4
(1 row)
```

- `lengthb(string)`

Description: Obtains the number of characters in a string. The value depends on character sets (GBK and UTF8).

Return type: integer

Example:

```
gaussdb=# SELECT lengthb('Chinese');
lengthb
-----
      7
(1 row)
```

- substr(string,from)

Description:

Extracts substrings from a string.

from indicates the start position of the extraction.

- If **from** starts at 0, the value **1** is used.
- If the value of **from** is positive, all characters from **from** to the end are extracted.
- If the value of **from** is negative, the last *n* characters in the string are extracted, in which **n** indicates the absolute value of **from**.

Return type: text

Example:

If the value of **from** is positive:

```
gaussdb=# SELECT substr('ABCDEF',2);
substr
-----
BCDEF
(1 row)
```

If the value of **from** is negative:

```
gaussdb=# SELECT substr('ABCDEF',-2);
substr
-----
EF
(1 row)
```

- substr(string,from,count)

Description:

Extracts substrings from a string.

from indicates the start position of the extraction.

count indicates the length of the extracted substring.

- If **from** starts at 0, the value **1** is used.
- If the value of **from** is positive, extract **count** characters starting from **from**.
- If the value of **from** is negative, extract the last **n count** characters in the string, in which **n** indicates the absolute value of **from**.
- If the value of **count** is smaller than **1**, **null** is returned.

Return type: text

Example:

If the value of **from** is positive:

```
gaussdb=# SELECT substr('ABCDEF',2,2);
substr
-----
BC
(1 row)
```

If the value of **from** is negative:

```
gaussdb=# SELECT substr('ABCDEF',-3,2);
substr
-----
DE
(1 row)
```

- **substrb(string,from)**

Description: The functionality of this function is the same as that of **SUBSTR(string,from)**. However, the calculation unit is byte.

Return type: text

Example:

```
gaussdb=# SELECT substrb('ABCDEF',-2);
substrb
-----
EF
(1 row)
```

- **substrb(string,from,count)**

Description: The functionality of this function is the same as that of **SUBSTR(string,from,count)**. However, the calculation unit is byte.

Return type: text

Example:

```
gaussdb=# SELECT substrb('ABCDEF',2,2);
substrb
-----
BC
(1 row)
```

- **to_single_byte(char)**

Description: Converts all multi-byte characters in a string to single-byte characters.

Return type: text

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- **to_multi_byte(char)**

Description: Converts all single-byte characters in a string to multi-byte characters.

Return type: text

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- **trim([leading |trailing |both] [characters] from string)**

Description: Removes the longest string containing only the characters (a space by default) from the start/end/both ends of the string.

Return type: text

Example:

```
gaussdb=# SELECT trim(BOTH 'x' FROM 'xTomxx');
btrim
-----
Tom
(1 row)
```

```
gaussdb=# SELECT trim(LEADING 'x' FROM 'xTomxx');
ltrim
-----
Tomxx
(1 row)
gaussdb=# SELECT trim(TRAILING 'x' FROM 'xTomxx');
rtrim
-----
xTom
(1 row)
```

- rtrim(string [, characters])

Description: Removes the longest string containing only characters from characters (a space by default) from the end of string.

Return type: text

Example:

```
gaussdb=# SELECT rtrim('TRIMxxxx','x');
rtrim
-----
TRIM
(1 row)
```

- ltrim(string [, characters])

Description: Removes the longest string containing only characters from characters (a space by default) from the start of string.

Return type: text

Example:

```
gaussdb=# SELECT ltrim('xxxxTRIM','x');
ltrim
-----
TRIM
(1 row)
```

- upper(string)

Description: Converts a character string into uppercase letters.

Return type: text

Example:

```
gaussdb=# SELECT upper('tom');
upper
-----
TOM
(1 row)
```

- lower(string)

Description: Converts a character string into lowercase letters.

Return type: text

Example:

```
gaussdb=# SELECT lower('TOM');
lower
-----
tom
(1 row)
```

- nls_upper(string [, nlsparam])

Description: Converts a character string to uppercase letters. You can specify a sorting rule to process special uppercase conversion rules in some languages. The format of **nlsparam** is '**nls_sort=sort_name**', where **sort_name** is replaced by the specific sorting rule name. When the **nlsparam** parameter is not set, this function is equivalent to **upper**.

Return type: text

Example:

```
gaussdb=# SELECT nls_upper('große');
 nls_upper
-----
GROßE
(1 row)
gaussdb=# SELECT nls_upper('große', 'nls_sort = XGerman');
 nls_upper
-----
GROSSE
(1 row)
```

 **NOTE**

This function can be used only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- `nls_lower(string [, nlsparam])`

Description: Converts a character string to lowercase letters. You can specify a sorting rule to process special lowercase conversion rules in some languages. The format of **nlsparam** is '**nls_sort=sort_name**', where **sort_name** is replaced by the specific sorting rule name. When the **nlsparam** parameter is not set, this function is equivalent to **lower**.

Return type: text

Example:

```
gaussdb=# SELECT nls_lower('INDIVISIBILITY');
 nls_lower
-----
indivisibility
(1 row)
gaussdb=# SELECT nls_lower('INDIVISIBILITY', 'nls_sort = XTurkish');
 nls_lower
-----
indivisibility
(1 row)
```

 **NOTE**

This function can be used only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- `instr(string,substring[,position,occurrence])`

Description: Queries and returns the value of the substring position that occurs the **occurrence** (1 by default) times from the **position** (1 by default) in the string.

- If the value of **position** is **0**, **0** is returned.
- If the value of **position** is negative, searches backwards from the last *n*th character in the string, in which *n* indicates the absolute value of **position**.

In this function, the calculation unit is character. One Chinese character is one character.

Return type: integer

Example:

```
gaussdb=# SELECT instr('corporate floor','or', 3);
 instr
-----
      5
(1 row)
```

```
gaussdb=# SELECT instr('corporate floor','or',-3,2);
instr
-----
      2
(1 row)
```

- `initcap(string)`

Description: Capitalizes the first letter of each word in a string.

Return type: text

Example:

```
gaussdb=# SELECT initcap('hi THOMAS');
initcap
-----
Hi Thomas
(1 row)
```

 **NOTE**

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2** in ORA-compatible database, if a case-insensitive character, such as Chinese, is followed by a case-sensitive character, only the first letter of the case-sensitive character is capitalized. Therefore, you are advised to set **a_format_version** to **10c** and **a_format_dev_version** to **s2**.

- `ascii(string)`

Description: Indicates the ASCII code of the first character in the string.

Return type: integer

Example:

```
gaussdb=# SELECT ascii('xyz');
ascii
-----
    120
(1 row)
```

- `ascii2(string)`

Description: Returns the decimal code of the first character of the input string in the database character set.

Return type: bigint

Example:

```
gaussdb=# SELECT ascii2('xyz');
ascii2
-----
    120
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- `asciistr(string)`

Description: Converts non-ASCII characters in the input string to `\XXXX`, where `XXXX` indicates the UTF-16 code unit.

Return type: text

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- `unistr(string)`

Description: Converts the coding sequence in a string into the corresponding character. Other characters remain unchanged.

Return type: text

 **NOTE**

- This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.
 - The backslash (\) must be followed by four hexadecimal characters to indicate the coding sequence, or another backslash (\) indicates that a single backslash (\) is entered.
 - If the input parameter is of the time type, the time type is implicitly converted to the character string type.
- `vsize(expr)`

Description: Returns the number of bytes of the input expression.

Return type: int

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- `replace(string varchar, search_string varchar, replacement_string varchar)`

Description: Replaces all **search_string** in the string with **replacement_string**.

Return type: text

Example:

```
gaussdb=# SELECT replace('jack and jue','j','bl');
         replace
-----
black and blue
(1 row)
```

- `concat(str1,str2)`

Description: Connects `str1` and `str2` and returns the string. Note: **concat** calls the output function of the data type and the return value is immutable. As a result, the optimizer cannot calculate the result in advance when generating a plan. If there are performance requirements, you are advised to use the operator `||`.

NOTICE

- If **sql_compatibility** is set to 'MYSQL' and **str1** or **str2** is set to **NULL**, the returned result is **NULL**.
- The return value of the `concat` function is of the variable-length type. When the `concat` function is compared with table data, the character string length is lost in the combination result. As a result, the comparison results are different.
- To prevent memory bloat, it is recommended that the number of nested layers of the function be less than or equal to five.

Return type: text

Example:

```
gaussdb=# SELECT concat('Hello', ' World!');
concat
-----
Hello World!
(1 row)
gaussdb=# SELECT concat('Hello', NULL);
concat
-----

(1 row)
gaussdb=# CREATE TABLE test_space(c char(10));
CREATE TABLE
gaussdb=# INSERT INTO test_space VALUES('a');
INSERT 0 1
-- After spaces are padded, the character string is still a fixed-length character string. It is expected
that the result can be found.
gaussdb=# SELECT * FROM test_space WHERE c = 'a ';
c
-----
a
(1 row)
-- The combination result is a variable-length character string. The comparison fails and the result
cannot be found.
gaussdb=# SELECT * FROM test_space WHERE c = 'a || ' ';
c
---
(0 rows)
```

- chr(integer)

Description: For the UTF-8 character set, the input is encoded as Unicode and a UTF-8 character is returned. For other character sets, an ASCII character is returned.

Return type: text

 NOTE

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1** in an ORA-compatible database, the value is truncated instead of being rounded off if the value of *integer* is a decimal.

Example:

```
gaussdb=# SELECT chr(65);
chr
----
A
(1 row)

-- If the UTF-8 character set is used (the database character set has been set to UTF-8 during
database creation, the database character set cannot be reset).
gaussdb=# SELECT chr(19968);
chr
----
—
(1 row)
```

- chr(cvalue int|bigint)

Description: Converts *cvalue* to the character of the corresponding byte order and returns the character.

cvalue can be converted into a value of the int or bigint type. The value range is $[0, 2^{32} - 1]$, corresponding to the range of unsigned int. A character array consisting of one to four bytes is returned based on the value of *n*. The byte arrays returned in different character sets are the same. However, due to

different encoding rules, the result of the returned character string varies depending on the character set encoding.

If the character set is a single-byte character set, an ASCII character is returned after *cvalue* mod 256.

Precautions:

- If a byte in the input *cvalue* is **0**, the output is truncated.
- If the input does not comply with the encoding rule of the current character set, an error is reported.
- If the input is **NULL** or **0**, **NULL** is returned.

Return type: text

Example:

```
gaussdb=# SELECT chr(65);
chr
-----
A
(1 row)
gaussdb=# CREATE DATABASE gaussdb_o WITH DBCOMPATIBILITY 'ORA';
gaussdb=# \c gaussdb_o
gaussdb_o=# SET a_format_version='10c';
gaussdb_o=# SET a_format_dev_version = 's1';
gaussdb_o=# SELECT chr(16705);
chr
-----
AA
(1 row)

-- The output is truncated.
gaussdb_o=# SELECT chr(4259905);
chr
-----
A
(1 row)
gaussdb_o=# \c postgres
gaussdb=# DROP DATABASE gaussdb_o;
```

NOTE

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1** in an ORA-compatible database, the `chr` function returns a character in the collation based on the input value. If the current character set for database encoding is a multi-byte character set, the return value contains one to four bytes. If the current character set for database encoding is a single-byte character set, the return value is a single byte obtained by performing the mod 256 operation on the input value. Otherwise, if the current character set for database encoding is UTF-8, the input is encoded as Unicode and a UTF-8 character is returned. For other character sets, an ASCII character is returned.

- `regexp_substr(source_char, pattern)`

Description: Extracts substrings from a regular expression. If the SQL syntax is ORA- and MySQL-compatible and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the period (.) cannot match the '\n' character. If **aformat_regexp_match** is not contained, the period (.) matches the '\n' character by default.

Return type: text

Example:

```
gaussdb=# SELECT regexp_substr('500 Hello World, Redwood Shores, CA', '[^,]+')
"REGEXPR_SUBSTR";
REGEXPR_SUBSTR
```

```
-----  
, Redwood Shores,  
(1 row)
```

- `regexp_replace(string, pattern, replacement [,flags])`

Description: Replaces substrings matching the POSIX-style regular expression. The source string is returned unchanged if there is no match to the pattern. If there is a match, the source string is returned with the replacement string substituted for the matching substring.

The replacement string can contain `\n`, where `n` is 1 through 9, to indicate that the source substring matching the `n`th parenthesized sub-expression of the pattern should be inserted, and it can contain `\&` to indicate that the substring matching the entire pattern should be inserted.

The optional **flags** argument contains zero or multiple single-letter flags that change the function behavior. The options supported by **flags** and description are shown in [Table 7-27](#).

Return type: varchar

Example:

```
gaussdb=# SELECT regexp_replace('Thomas', '[mN]a.', 'M');  
regexp_replace  
-----  
ThM  
(1 row)  
gaussdb=# SELECT regexp_replace('foobarbaz','b(..)', E'X\|Y', 'g') AS  
RESULT;  
result  
-----  
fooXarYXazY  
(1 row)
```

- `regexp_replace(string text, pattern text [, replacement text [, position int [, occurrence int [, flags text]]]])`

Description: Replaces substrings matching the POSIX-style regular expression. The source string is returned unchanged if there is no match to the pattern. If there is a match, the source string is returned with the replacement string substituted for the matching substring.

Parameters:

- **string**: source character string used for matching.
- **pattern**: regular expression pattern string used for matching.
- **replacement**: character string used to replace the matched substring. This parameter is optional. If no parameter value is specified or the parameter value is null, an empty string is used for replacement.
- **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
- **occurrence**: sequence number of the matched substring to be replaced. This parameter is optional. The default value is **1**, indicating that the first matched substring is replaced.
- **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. The options supported by **flags** and description are shown in [Table 7-27](#).

 NOTE

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1** in an ORA-compatible database, the default value of **occurrence** is **0**, indicating that all matched substrings are replaced and the pattern parameter ending with a backslash (\) is valid.

Return type: text

Example:

```
gaussdb=# SELECT regexp_replace('foobarbaz','b(..)', 'E'\1Y', 2, 2, 'n') AS RESULT;
result
-----
foobarXazY
(1 row)
```

- `concat_ws(sep text, str"any" [, str"any" [, ...]])`

Description: Uses the first parameter as the separator, which is associated with all following parameters. The **NULL** parameter is ignored.

NOTICE

- If the first parameter value is **NULL**, the returned result is **NULL**.
- If the first parameter is provided but the parameter value is an empty string (") and the SQL compatibility mode of the database is set to ORA, the returned result is **NULL**. This is because the ORA-compatible mode treats the empty string (") as **NULL**. To resolve this problem, you can change the SQL compatibility mode of the database to MySQL, TD, or PG.
- To prevent memory bloat, it is recommended that the number of nested layers of the function be less than or equal to five.

Return type: text

Example:

```
gaussdb=# SELECT concat_ws(',', 'ABCDE', 2, NULL, 22);
concat_ws
-----
ABCDE,2,22
(1 row)
```

- `nlssort(string text, sort_method text)`

Description: Returns the encoding value of a string in the sorting mode specified by **sort_method**. The encoding value can be used for sorting and determines the sequence of the string in the sorting mode. Currently, **sort_method** can be set to **nls_sort=schinese_pinyin_m** or **nls_sort=generic_m_ci**. **nls_sort=generic_m_ci** supports only the case-insensitive order for English characters.

String type: text

sort_method type: text

Return type: text

Example:

```
gaussdb=# CREATE TABLE test(a text);
gaussdb=# INSERT INTO test(a) VALUES('abc');
gaussdb=# INSERT INTO test(a) VALUES('abc');
gaussdb=# INSERT INTO test(a) VALUES('abc');
gaussdb=# SELECT * FROM test ORDER BY nlssort(a,'nls_sort=schinese_pinyin_m');
```

```

a
-----
abc
abC
abC
(3 rows)
gaussdb=# SELECT * FROM test ORDER BY nlssort(a, 'nls_sort=generic_m_ci');
a
-----
abC
abc
abC
(3 rows)
gaussdb=# DROP TABLE test;

```

- `convert(string bytea, src_encoding name, dest_encoding name)`

Description: Converts the string to **dest_encoding**. **src_encoding** specifies the source code encoding. The string must be valid in this encoding.

Return type: bytea

Example:

```

gaussdb=# SELECT convert('text_in_utf8', 'UTF8', 'GBK');
convert
-----
\x746578745f696e5f75746638
(1 row)

```

NOTE

If the rule for converting between source to target encoding (for example, GBK and LATIN1) does not exist, the string is returned without conversion. See the `pg_conversion` system catalog for details.

Example:

```

gaussdb=# SHOW server_encoding;
server_encoding
-----
LATIN1
(1 row)

gaussdb=# SELECT convert_from('some text', 'GBK');
convert_from
-----
some text
(1 row)

db_latin1=# SELECT convert_to('some text', 'GBK');
convert_to
-----
\x736f6d652074657874
(1 row)

db_latin1=# SELECT convert('some text', 'GBK', 'LATIN1');
convert
-----
\x736f6d652074657874
(1 row)

```

- `convert_from(string bytea, src_encoding name)`

Description: Converts a string using the coding mode of the database.

src_encoding specifies the source code encoding. The string must be valid in this encoding.

Return type: text

Example:

```
gaussdb=# SELECT convert_from('text_in_utf8', 'UTF8');
convert_from
-----
text_in_utf8
(1 row)
```

- `convert_to(string text, dest_encoding name)`
Description: Converts a string to **dest_encoding**.
Return type: bytea

Example:

```
gaussdb=# SELECT convert_to('some text', 'UTF8');
convert_to
-----
\x736f6d652074657874
(1 row)
```

- `string [NOT] LIKE pattern [ESCAPE escape-character]`

Description: Specifies the pattern matching function.

If the pattern does not include a percentage sign (%) or an underscore (_), this mode represents itself only. In this case, the behavior of LIKE is the same as the equal operator. The underscore (_) in the pattern matches any single character while one percentage sign (%) matches no or multiple characters.

To match with underscores (_) or percent signs (%), corresponding characters in **pattern** must lead escape characters. The default escape character is a backward slash (\) and can be specified using the **ESCAPE** clause. To match with escape characters, enter two escape characters.

Return type: Boolean

Example:

```
gaussdb=# SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
result
-----
f
(1 row)
gaussdb=# SELECT 'AA@_BBCC' LIKE '%A@_B%' AS RESULT;
result
-----
t
(1 row)
```

- `REGEXP_LIKE(source_string, pattern [, match_parameter])`

Description: Indicates the mode matching function of a regular expression.

source_string indicates the source string and **pattern** indicates the matching pattern of the regular expression. **match_parameter** indicates the matching items and the values are as follows:

- 'i': case-insensitive
- 'c': case-sensitive
- 'n': allowing the metacharacter "." in a regular expression to be matched with a linefeed.
- 'm': allows **source_string** to be regarded as multiple rows.

If **match_parameter** is ignored, **case-sensitive** is enabled by default, "." is not matched with a linefeed, and **source_string** is regarded as a single row.

Return type: Boolean

Example:

```
gaussdb=# SELECT regexp_like('ABC', '[A-Z]');
regexp_like
-----
t
(1 row)
gaussdb=# SELECT regexp_like('ABC', '[D-Z]');
regexp_like
-----
f
(1 row)
gaussdb=# SELECT regexp_like('ABC', '[A-Z]', 'i');
regexp_like
-----
t
(1 row)
gaussdb=# SELECT regexp_like('ABC', '[A-Z]');
regexp_like
-----
t
(1 row)
```

- `format(formatstr text [, str"any" [, ...]])`

Description: Formats a string.

Return type: text

Example:

```
gaussdb=# SELECT format('Hello %s, %1$s', 'World');
format
-----
Hello World, World
(1 row)
```

 **NOTE**

To prevent memory bloat, it is recommended that the number of nested layers of the function be less than or equal to five.

- `md5(string)`

Description: Encrypts a string in MD5 mode and returns a value in hexadecimal form.

 **NOTE**

The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.

Return type: text

Example:

```
gaussdb=# SELECT md5('ABC');
md5
-----
902fbbd2b1df0c4f70b4a5d23525e932
(1 row)
```

- `sha(string) / sha1(string)`

Description: Encrypts a string using SHA1 and returns a hexadecimal number. The sha and sha1 functions are the same.

 **NOTE**

- The SHA1 encryption algorithm is not recommended because it has lower security and poses security risks.
- This function is valid only when GaussDB is compatible with MySQL (that is, **sql_compatibility** is set to 'MYSQL').

Return type: text

Example:

```
gaussdb=# SELECT sha('ABC');
          sha
-----
3c01bdbb26f358bab27f267924aa2c9a03fcfdb8
(1 row)
gaussdb=# SELECT sha1('ABC');
          sha1
-----
3c01bdbb26f358bab27f267924aa2c9a03fcfdb8
(1 row)
```

- sha2(string, hash_length)

Description: Encrypts a string in SHA-2 mode and returns a value in hexadecimal form.

hash_length: corresponds to an SHA2 algorithm. The value can be **0** (SHA-256), **224** (SHA-224), **256** (SHA-256), **384** (SHA-384), or **512** (SHA-512). For other values, **NULL** is returned.

 **NOTE**

- The SHA-224 encryption algorithm is not recommended because it has lower security and poses security risks.
- The SHA2 function records hash plaintext in logs. Therefore, you are advised not to use this function to encrypt sensitive information such as keys.
- This function is valid only when GaussDB is compatible with MySQL (that is, **sql_compatibility** is set to 'MYSQL').

Return type: text

Example:

```
gaussdb=# SELECT sha2('ABC',224);
          sha2
-----
107c5072b799c4771f328304cfe1ebb375eb6ea7f35a3aa753836fad
(1 row)
gaussdb=# SELECT sha2('ABC',256);
          sha2
-----
b5d4045c3f466fa91fe2cc6abe79232a1a57cdf104f7a26e716e0a1e2789df78
(1 row)
gaussdb=# SELECT sha2('ABC',0);
          sha2
-----
b5d4045c3f466fa91fe2cc6abe79232a1a57cdf104f7a26e716e0a1e2789df78
(1 row)
```

- decode(string text, format text)

Description: Decodes binary data from textual representation.

Return type: bytea

Example:

```
gaussdb=# SELECT decode('MTIzAAE=', 'base64');
          decode
```

```
-----
\x3132330001
(1 row)
```

- `similar_escape(pat text, esc text)`

Description: Converts a regular expression of the SQL:2008 style to the POSIX style.

Return type: text

Example:

```
gaussdb=# select similar_escape('\s+ab','2');
similar_escape
-----
^(?:\s+ab)$
(1 row)
```

- `find_in_set(text, set)`

Description: Finds the position of a given member in a set, counting from 1. If no record is found, 0 is returned. The distributed system does not support the SET data type. An error will be reported when this function is executed.

Return type: int2

- `encode(data bytea, format text)`

Description: Encodes binary data into a textual representation.

Return type: text

Example:

```
gaussdb=# SELECT encode(E'123\000\001', 'base64');
encode
-----
MTIzAAE=
(1 row)
```

NOTE

- For a string containing newline characters, for example, a string consisting of a newline character and a space, the value of **length** and **lengthb** in GaussDB is 2.
- In GaussDB, *n* in the CHAR(*n*) type indicates the number of characters. Therefore, for multiple-octet coded character sets, the length returned by the LENGTHB function may be longer than *n*.
- GaussDB supports multiple types of databases, including ORA, MySQL, TD, and PG. The lexical analyzer of ORA-compatible database is different from that of the other three databases. In an ORA-compatible database, an empty string is considered as **NULL**. Therefore, when an ORA-compatible database is used, if an empty string is used as a parameter in the preceding character operation function, no output is displayed.

Example:

```
gaussdb=# SELECT translate('12345','123','');
translate
-----
(1 row)
```

This is because the kernel checks whether the input parameter contains **NULL** before calling the corresponding function. If the input parameter contains **NULL**, the kernel does not call the corresponding function. As a result, no output is displayed. In PG-compatible mode, the processing of character strings is the same as that of PG. Therefore, the preceding problem does not occur.

Extension Functions and Operators

- `pkg_bpchar_opc()`

Description: Serves as an extension API to add the comparison operator between bpchar and text or between text and bpchar policies, so as to solve

the problem that indexes cannot be matched when data of the bpchar and text types is compared. Only system administrators can install extensions.

NOTICE

The extended function is for internal use only. You are advised not to use it.

Example:

The tables **logs_nchar**, **logs_char**, **logs_varchar2**, and **logs_text** are as follows:

```
/*
logs_nchar table
*/
DROP TABLE IF EXISTS logs_nchar;
CREATE TABLE logs_nchar
(
  log_id    NCHAR(16),
  log_message VARCHAR2(255) NOT NULL
);
DROP INDEX IF EXISTS idx_nchar_logid;
CREATE INDEX idx_nchar_logid ON logs_nchar(log_id);
INSERT INTO logs_nchar VALUES('FE306991300002 ', '111');
INSERT INTO logs_nchar VALUES('FE306991300003 ', '222');
INSERT INTO logs_nchar VALUES('FE306991300004 ', '222');

/*
logs_char table
*/
DROP TABLE IF EXISTS logs_char;
CREATE TABLE logs_char
(
  log_id    CHAR(16),
  log_message VARCHAR2(255) NOT NULL
);
DROP INDEX IF EXISTS idx_char_logid;
CREATE INDEX idx_char_logid ON logs_char(log_id);
INSERT INTO logs_char VALUES('FE306991300002 ', '111');
INSERT INTO logs_char VALUES('FE306991300003 ', '222');
INSERT INTO logs_char VALUES('FE306991300004 ', '222');

/*
logs_varchar2 table
*/
DROP TABLE IF EXISTS logs_varchar2;
CREATE TABLE logs_varchar2
(
  log_id    VARCHAR2(16),
  log_message VARCHAR2(255) NOT NULL
);
DROP INDEX IF EXISTS idx_varchar2_logid;
CREATE INDEX idx_varchar2_logid ON logs_varchar2(log_id);
INSERT INTO logs_varchar2 VALUES('FE306991300002 ', '111');
INSERT INTO logs_varchar2 VALUES('FE306991300003 ', '222');
INSERT INTO logs_varchar2 VALUES('FE306991300004 ', '222');

/*
logs_text table
*/
DROP TABLE IF EXISTS logs_text;
CREATE TABLE logs_text
(
  log_id    text,
```

```

log_message VARCHAR2(255) NOT NULL
);
DROP INDEX IF EXISTS idx_text_logid;
CREATE INDEX idx_text_logid ON logs_text(log_id);
INSERT INTO logs_text VALUES('FE306991300002 ', '111');
INSERT INTO logs_text VALUES('FE306991300003 ', '222');
INSERT INTO logs_text VALUES('FE306991300004 ', '222');

```

Compare the bpchar type with the text type (initial state, forward compatibility).

```

/*
The purpose is to obtain the detailed execution plan of all nodes.
*/
gaussdb=# SET max_datanode_for_plan = 64;
SET
/*
If no extension is installed, when nchar and text are compared, nchar is implicitly converted to text
because there is no bpchar or text index operator. That is, the fixed-length character type is converted to
the variable-length character type. As a result, the execution plan changes and the index cannot be
matched.
*/
gaussdb=# EXPLAIN SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002 '),16,
');

```

QUERY PLAN

```

-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
Node/s: All datanodes

Remote SQL: SELECT log_id, log_message FROM public.logs_nchar WHERE log
_id::text = rpad(btrim('FE306991300002 '::text), 16, '::text)
Datanode Name: datanode1
Seq Scan on logs_nchar (cost=0.00..1.01 rows=1 width=584)
Filter: ((log_id)::text = 'FE306991300002 '::text)

Datanode Name: datanode2
Seq Scan on logs_nchar (cost=0.00..1.03 rows=1 width=584)
Filter: ((log_id)::text = 'FE306991300002 '::text)

```

```

(12 rows)
/*
The log_id column in the logs_nchar table is of nchar(16) type. The inserted data is
'FE306991300002 ', which is implicitly converted to text. During comparison, spaces are deleted, that
is, 'FE306991300002'='FE306991300002 '. Therefore, the data is not matched.
*/
gaussdb=# SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002 '),16, ' ');
log_id | log_message
-----+-----
(0 rows)

```

Compare the bpchar type with the text type (the pkg_bpchar_opc extension is installed, which is ORA-compatible).

```

/*
A system administrator installs the pkg_bpchar_opc extension. The comparison operators of the
bpchar and text types and index-related content are added to the database.
*/
gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION

gaussdb=# SET max_datanode_for_plan = 64;
SET
gaussdb=# EXPLAIN SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002 '),16,
');

```

QUERY PLAN

```

-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
Node/s: datanode2

```

```
Remote SQL: SELECT log_id, log_message FROM public.logs_nchar WHERE log_id = rpad(btrim('FE306991300002 '::text), 16, ' '::text)
Datanode Name: datanode2
[Bypass]
Index Scan using idx_nchar_logid on logs_nchar (cost=0.00..8.27 rows=1 width=584)
Index Cond: (log_id = 'FE306991300002 '::text)

(9 rows)
/*
In this case, when log_id is implicitly converted to the bpchar type and compared with the text type, the comparison operator and index information can be found, and the index can be matched.
*/
gaussdb=# SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002 '),16,' ');
log_id | log_message
-----+-----
FE306991300002 | 111
(1 row)
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
DROP EXTENSION
```

Compare the text type with the bpchar type (initial state, forward compatibility).

```
gaussdb=# SET max_datanode_for_plan = 64;
SET
gaussdb=# EXPLAIN SELECT * FROM logs_text WHERE log_id = 'FE306991300002 '::bpchar;
QUERY PLAN

-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
Node/s: datanode2

Remote SQL: SELECT log_id, log_message FROM public.logs_text WHERE log_id = 'FE306991300002 '::bpchar::text
Datanode Name: datanode2
[Bypass]
Index Scan using idx_text_logid on logs_text (cost=0.00..8.27 rows=1 width=548)
Index Cond: (log_id = 'FE306991300002'::text)

(9 rows)
gaussdb=# SELECT * FROM logs_text WHERE log_id = 'FE306991300002 '::bpchar;
log_id | log_message
-----+-----
(0 rows)
gaussdb=# SELECT * FROM logs_text WHERE log_id = 'FE306991300002 '::text;
log_id | log_message
-----+-----
FE306991300002 | 111
(1 row)
```

Compare the text type with the bpchar type (the pkg_bpchar_opc extension is installed, which is ORA-compatible).

```
gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION
gaussdb=# SET max_datanode_for_plan = 64;
SET
gaussdb=# explain select * from logs_text t1 where t1.log_id ='FE306991300002 '::bpchar;
QUERY PLAN

-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
Node/s: datanode2

Remote SQL: SELECT log_id, log_message FROM public.logs_text t1 WHERE log_id = 'FE306991300002 '::bpchar
Datanode Name: datanode2
[Bypass]
```

```

Index Scan using idx_text_logid on logs_text t1 (cost=0.00..8.27 row
s=1 width=548)
  Index Cond: (log_id = 'FE306991300002 '::bpchar)

(9 rows)
gaussdb=# SELECT * FROM logs_text WHERE log_id = 'FE306991300002 '::bpchar;
 log_id | log_message
-----+-----
(0 rows)
gaussdb=# SELECT * FROM logs_text WHERE log_id = 'FE306991300002 '::text;
 log_id | log_message
-----+-----
FE306991300002 | 111
(1 row)
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
DROP EXTENSION

```

Compare the text type with the bpchar type (initial state, forward compatibility).

```

gaussdb=# SET max_datanode_for_plan = 64;
SET
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
 log_id | log_message | log_id | log_message
-----+-----+-----+-----
FE306991300003 | 222 | FE306991300003 | 222
FE306991300004 | 222 | FE306991300004 | 222
FE306991300002 | 111 | FE306991300002 | 111
(3 rows)
gaussdb=# EXPLAIN SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
QUERY PLAN

```

```

-----
Streaming (type: GATHER) (cost=17.29..39.84 rows=20 width=1150)
  Node/s: All datanodes
  -> Hash Join (cost=13.29..31.42 rows=20 width=1150)
    Hash Cond: (((t1.log_id)::bpchar) = t2.log_id)
    -> Streaming(type: REDISTRIBUTE) (cost=0.00..17.98 rows=20 width=566)
    Spawn on: All datanodes
    -> Seq Scan on logs_varchar2 t1 (cost=0.00..13.13 rows=20 width=566)
    -> Hash (cost=13.13..13.13 rows=21 width=584)
      -> Seq Scan on logs_char t2 (cost=0.00..13.13 rows=20 width=584)
(9 rows)
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002 '::text;
 log_id | log_message | log_id | log_message
-----+-----+-----+-----
FE306991300002 | 111 | FE306991300003 | 222
FE306991300002 | 111 | FE306991300002 | 111
FE306991300002 | 111 | FE306991300004 | 222
(3 rows)

```

Compare the text type with the bpchar type (the pkg_bpchar_opc extension is installed, which is ORA-compatible).

```

gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION
gaussdb=# SET max_datanode_for_plan = 64;
SET

```

```

/*
This format is not recommended. After the extension is installed, the varchar2 type of log_id in the t1
table is implicitly converted to the text type. When it is compared with the log_id in the t2 table, the
char type of log_id in the t2 table is implicitly converted to the bpchar type. In this case, spaces after
log_id is removed by the database, that is, 'FE306991300002'='FE306991300002 '. Therefore, no
data is matched.
*/

```

```

/*
Incorrect example:

```

```

*/
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
 log_id | log_message | log_id | log_message
-----+-----+-----+-----
(0 rows)

gaussdb=# EXPLAIN SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
QUERY
PLAN
-----
Streaming (type: GATHER) (cost=17.29..45.25 rows=20 width=1150)
 Node/s: All datanodes
  -> Hash Join (cost=13.29..36.83 rows=20 width=1150)
      Hash Cond: ((t1.log_id)::text = t2.log_id)
      -> Streaming(type: BROADCAST) (cost=0.00..23.37 rows=40 width
=566)
          Spawn on: All datanodes
          -> Seq Scan on logs_varchar2 t1 (cost=0.00..13.13 rows=
20 width=566)
              -> Hash (cost=13.13..13.13 rows=21 width=584)
                  -> Seq Scan on logs_char t2 (cost=0.00..13.13 rows=20 w
idth=584)
(9 rows)

gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002 ';
 log_id | log_message | log_id | log_message
-----+-----+-----+-----
FE306991300002 | 111 | FE306991300002 | 111
FE306991300002 | 111 | FE306991300003 | 222
FE306991300002 | 111 | FE306991300004 | 222
(3 rows)

/*
This format is recommended to avoid the following problems: the data type of log_id in the t1 table
is converted to the text type, spaces are reserved during comparison, and data cannot be matched
when the data type of log_id in the t2 table is compared with the data type of log_id in the t1 table.
The type of t1 table is forcibly converted to the bpchar type before the extension is installed, that is,
'FE306991300002' = 'FE306991300002'. Therefore, data is matched.
*/
/*
Correct example:
*/
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id::bpchar = t2.log_id;
 log_id | log_message | log_id | log_message
-----+-----+-----+-----
FE306991300002 | 111 | FE306991300002 | 111
FE306991300004 | 222 | FE306991300004 | 222
FE306991300003 | 222 | FE306991300003 | 222
(3 rows)
gaussdb=# EXPLAIN SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id::bpchar =
t2.log_id;
QUERY PLAN
-----
Streaming (type: GATHER) (cost=17.29..45.25 rows=20 width=1150)
 Node/s: All datanodes
  -> Hash Join (cost=13.29..36.83 rows=20 width=1150)
      Hash Cond: ((t1.log_id)::bpchar = t2.log_id)
      -> Streaming(type: BROADCAST) (cost=0.00..23.37 rows=40 width
=566)
          Spawn on: All datanodes
          -> Seq Scan on logs_varchar2 t1 (cost=0.00..13.13 rows=
20 width=566)
              -> Hash (cost=13.13..13.13 rows=21 width=584)
                  -> Seq Scan on logs_char t2 (cost=0.00..13.13 rows=20 w
idth=584)

```

```
(9 rows)
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002 ';
 log_id | log_message | log_id | log_message
-----+-----+-----+-----
FE306991300002 | 111 | FE306991300002 | 111
FE306991300002 | 111 | FE306991300003 | 222
FE306991300002 | 111 | FE306991300004 | 222
(3 rows)
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
DROP EXTENSION

gaussdb=# DROP TABLE logs_nchar;
gaussdb=# DROP TABLE logs_char;
gaussdb=# DROP TABLE logs_varchar2;
gaussdb=# DROP TABLE logs_text;
```

 **NOTE**

- This solves the problem that data and indexes cannot be properly matched when equality matching is performed between the bpchar type (containing multiple spaces) and the text type.
- The UB-tree and B-tree are involved. The comparison symbols include >, >=, <, <=, and <>.
- The impact scope involves implicit conversion between character types. For example, when a variable-length data type is compared with a fixed-length data type, the variable-length data type is preferentially converted to the text type instead of the original bpchar type.
- The pkg_bpchar_opc extension is disabled by default. You can check whether the extension is enabled in the pg_extension system catalog. If the extension data exists, the extension is enabled. If the extension data does not exist, the extension is disabled. When extension is disabled, forward compatibility is maintained. When extension is enabled, compatibility with database ORA is maintained. After the pkg_bpchar_opc extension is enabled, you need to set the **max_datanode_for_plan** parameter as required. This parameter specifies the number of execution plans displayed on DNs when the FQS plan is generated. For example, to set **max_datanode_for_plan** to **64**, run the following command:
set max_datanode_for_plan = 64

Table 7-28 Functions supported by pkg_bpchar_opc

Name	Description
pg_catalog.bpchar_text_lt	Compares the bpchar type with the text type to check whether the value on the left is less than the value on the right.
pg_catalog.bpchar_text_le	Compares the bpchar type with the text type to check whether the value on the left is less than or equal to the value on the right.
pg_catalog.bpchar_text_eq	Compares the bpchar type with the text type to check whether the value on the left is equal to the value on the right.
pg_catalog.bpchar_text_ge	Compares the bpchar type with the text type to check whether the value on the left is greater than or equal to the value on the right.

Name	Description
pg_catalog.bpchar_text_gt	Compares the bpchar type with the text type to check whether the value on the left is greater than the value on the right.
pg_catalog.bpchar_text_ne	Compares the bpchar type with the text type and checks whether the value on the left is different from the value on the right.
pg_catalog.bpchar_text_cmp	Specifies that the index of the bpchar and text types supports comparison functions.
pg_catalog.text_bpchar_lt	Compares the text type with the bpchar type to check whether the value on the left is less than the value on the right.
pg_catalog.text_bpchar_le	Compares the text type with the bpchar type to check whether the value on the left is less than or equal to the value on the right.
pg_catalog.text_bpchar_eq	Compares the text type with the bpchar type to check whether the value on the left is equal to the value on the right.
pg_catalog.text_bpchar_ge	Compares the text type with the bpchar type to check whether the value on the left is greater than or equal to the value on the right.
pg_catalog.text_bpchar_gt	Compares the text type with the bpchar type to check whether the value on the left is greater than the value on the right.
pg_catalog.text_bpchar_ne	Compares the text type with the bpchar type and checks whether the value on the left is different from the value on the right.
pg_catalog.text_bpchar_cmp	Specifies that the index of the text and bpchar types supports comparison functions.
pg_catalog.hashbpchar_text	Specifies that the hash of the bpchar and text types supports comparison functions.
pg_catalog.hashtextbpchar	Specifies that the hash of the text and bpchar types supports comparison functions.

7.5.4 Binary String Functions and Operators

String Operators

SQL defines some string functions that use keywords, rather than commas, to separate arguments.

- `octet_length(string)`

Description: Specifies the number of bytes in a binary string.

Return type: int

Example:

```
gaussdb=# SELECT octet_length(E'jo\00se'::bytea) AS RESULT;
result
-----
      5
(1 row)
```

- `overlay(string placing string from int [for int])`

Description: Replaces substrings.

Return type: bytea

Example:

```
gaussdb=# SELECT overlay(E'Th\00omas'::bytea placing E'\002\003'::bytea from 2 for 3) AS
RESULT;
result
-----
\x5402036d6173
(1 row)
```

- `position(substring in string)`

Description: Specifies the location of the specified substring.

Return type: int

Example:

```
gaussdb=# SELECT position(E'\00om'::bytea in E'Th\00omas'::bytea) AS RESULT;
result
-----
      3
(1 row)
```

- `substring(string [from int] [for int])`

Description: Truncates substrings.

Return type: bytea

Example:

```
gaussdb=# SELECT substring(E'Th\00omas'::bytea from 2 for 3) AS RESULT;
result
-----
\x68006f
(1 row)
```

- `substr(bytea [from int] [for int])`

Description: Truncates substrings.

Return type: bytea

Example:

```
gaussdb=# SELECT substr(E'Th\00omas'::bytea,2, 3) as result;
result
-----
\x68006f
(1 row)
```

- `trim([both] bytes from string)`

Description: Removes the longest string containing only bytes from **bytes** from the start and end of **string**.

Return type: bytea

Example:

```
gaussdb=# SELECT trim(E'\000'::bytea from E'\000Tom\000'::bytea) AS RESULT;
result
-----
```

```
\x546f6d  
(1 row)
```

Other Binary String Functions

GaussDB provides common syntax used for calling functions.

- `btrim(string bytea, bytes bytea)`
Description: Removes the longest string containing only bytes from **bytes** from the start and end of **string**.

Return type: bytea

Example:

```
gaussdb=# SELECT btrim(E'\000trim\000'::bytea, E'\000'::bytea) AS RESULT;  
result  
-----  
\x7472696d  
(1 row)
```

- `get_bit(string, offset)`
Description: Extracts bits from a string.

Return type: int

Example:

```
gaussdb=# SELECT get_bit(E'Th\000omas'::bytea, 45) AS RESULT;  
result  
-----  
1  
(1 row)
```

- `get_byte(string, offset)`
Description: Extracts bytes from a string.

Return type: int

Example:

```
gaussdb=# SELECT get_byte(E'Th\000omas'::bytea, 4) AS RESULT;  
result  
-----  
109  
(1 row)
```

- `set_bit(string, offset, newvalue)`
Description: Sets bits in a string.

Return type: bytea

Example:

```
gaussdb=# SELECT set_bit(E'Th\000omas'::bytea, 45, 0) AS RESULT;  
result  
-----  
\x5468006f6d4173  
(1 row)
```

- `set_byte(string, offset, newvalue)`
Description: Sets bytes in a string.

Return type: bytea

Example:

```
gaussdb=# SELECT set_byte(E'Th\000omas'::bytea, 4, 64) AS RESULT;  
result  
-----  
\x5468006f406173  
(1 row)
```

- rawcmp
Description: Specifies the raw data type comparison function.
Parameter: raw, raw
Return type: integer
- raweq
Description: Specifies the raw data type comparison function.
Parameter: raw, raw
Return type: Boolean
- rawge
Description: Specifies the raw data type comparison function.
Parameter: raw, raw
Return type: Boolean
- rawgt
Description: Specifies the raw data type comparison function.
Parameter: raw, raw
Return type: Boolean
- rawin
Description: Specifies the raw data type parsing function.
Parameter:cstring
Return type:bytea
- rawle
Description: Specifies the raw data type parsing function.
Parameter: raw, raw
Return type: Boolean
- rawlike
Description: Specifies the raw data type parsing function.
Parameter: raw, raw
Return type: Boolean
- rawlt
Description: Specifies the raw data type parsing function.
Parameter: raw, raw
Return type: Boolean
- rawne
Description: Compares whether the raw types are the same.
Parameter: raw, raw
Return type: Boolean
- rawnlike
Description: Checks whether the raw type matches the mode.
Parameter: raw, raw
Return type: Boolean

- **rawout**
Description: Specifies the RAW output API.
Parameter: bytea
Return type: cstring
- **rawsend**
Description: Converts bytea to the binary type.
Parameter: raw
Return type: bytea
- **rawtohex**
Description: Converts the raw format to the hexadecimal format.
Parameter: text
Return type: text

7.5.5 Bit String Functions and Operators

Bit String Operators

Aside from the usual comparison operators, the following operators can be used. Bit string operands of **&**, **|**, and **#** must be of equal length. In case of bit shifting, the original length of the string is preserved by zero padding (if necessary).

- **||**
Description: Connects bit strings.

Example:

```
gaussdb=# SELECT B'10001' || B'011' AS RESULT;
result
-----
10001011
(1 row)
```

NOTE

- A field can have a maximum of 180 consecutive internal joins. A field with excessive joins will be split into joined consecutive strings.
Example: **str1||str2||str3||str4** is split into **(str1||str2)||(str3||str4)**.
- In ORA-compatible mode, if bit strings contain a null string, the null string is ignored and other strings are joined. In other compatibility modes, the null string is returned.
Take **str1||NULL||str2** as an example. **str1str2** is returned in ORA-compatible mode and **NULL** is returned in other compatibility modes.

- **&**
Description: Specifies the AND operation between bit strings.

Example:

```
gaussdb=# SELECT B'10001' & B'01101' AS RESULT;
result
-----
00001
(1 row)
```

- **|**
Description: Specifies the OR operation between bit strings.

Example:

```
gaussdb=# SELECT B'10001' | B'01101' AS RESULT;
result
-----
11101
(1 row)
```

- #

Description: Specifies the OR operation between bit strings if they are inconsistent. If the same positions in the two bit strings are both 1 or 0, the position returns **0**.

Example:

```
gaussdb=# SELECT B'10001' # B'01101' AS RESULT;
result
-----
11100
(1 row)
```

- ~

Description: Specifies the NOT operation between bit strings.

Example:

```
gaussdb=# SELECT ~B'10001' AS RESULT;
result
-----
01110
(1 row)
```

- <<

Description: Shifts left in a bit string.

Example:

```
gaussdb=# SELECT B'10001' << 3 AS RESULT;
result
-----
01000
(1 row)
```

- >>

Description: Shifts right in a bit string.

Example:

```
gaussdb=# SELECT B'10001' >> 2 AS RESULT;
result
-----
00100
(1 row)
```

The following SQL-standard functions work on bit strings as well as strings: **length**, **bit_length**, **octet_length**, **position**, **substring**, and **overlay**.

The following functions work on bit strings as well as binary strings: **get_bit** and **set_bit**. When working with a bit string, these functions number the first (leftmost) bit of the string as bit 0.

In addition, it is possible to convert between integral values and type **bit**. Example:

```
gaussdb=# SELECT 44::bit(10) AS RESULT;
result
-----
0000101100
(1 row)

gaussdb=# SELECT 44::bit(3) AS RESULT;
result
```

```
-----
100
(1 row)

gaussdb=# SELECT cast(-44 as bit(12)) AS RESULT;
      result
-----
111111010100
(1 row)

gaussdb=# SELECT '1110'::bit(4)::integer AS RESULT;
      result
-----
      14
(1 row)

gaussdb=# select substring('10101111'::bit(8), 2);
 substring
-----
0101111
(1 row)
```

 NOTE

Casting to just "bit" means casting to bit(1), and so will deliver only the least significant bit of the integer.

7.5.6 Pattern Matching Operators

The database provides three independent methods for implementing pattern matching: SQL LIKE operator, SIMILAR TO operator, and POSIX-style regular expressions. Besides these basic operators, functions can be used to extract or replace matching substrings and to split a string at matching locations.

- LIKE

Description: Specifies whether the string matches the pattern string following LIKE. The LIKE expression returns true if the string matches the supplied pattern. (As expected, the NOT LIKE expression returns false if LIKE returns true, and vice versa.)

Matching rules:

- This operator can succeed only when its pattern matches the entire string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign.
- The underscore (_) represents (matching) any single character. Percentage (%) indicates the wildcard character of any string.
- To match a literal underscore or percent sign, the respective character in pattern must be preceded by the escape character. The default escape character is one backslash but a different one can be selected by using the ESCAPE clause.
- To match with escape characters, enter two escape characters. For example, to write a pattern constant containing a backslash (\), you need to enter two backslashes in SQL statements.

 NOTE

When **standard_conforming_strings** is set to **off**, any backslashes you write in literal string constants will need to be doubled. Therefore, writing a pattern matching a single backslash is actually going to write four backslashes in the statement. You can avoid this by selecting a different escape character by using **ESCAPE**, so that the backslash is no longer a special character of **LIKE**. But the backslash is still the special character of the character text analyzer. In this case, two backslashes are required. You can also select no escape character by writing **ESCAPE ''**. This effectively disables the escape mechanism, but it does not eliminate the special meaning of underscore and percent signs in the pattern.

- e. The keyword **ILIKE** can be used instead of **LIKE** to make the match case-insensitive.
- f. Operator **~~** is equivalent to **LIKE**, and operator **~~*** corresponds to **ILIKE**.

Example:

```
gaussdb=# SELECT 'abc' LIKE 'abc' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' LIKE 'a%' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' LIKE '_b_' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' LIKE 'c' AS RESULT;
result
-----
f
(1 row)
```

- **SIMILAR TO**

Description: Returns true or false depending on whether the pattern matches the given string. It is similar to **LIKE**, but differs in that it uses the regular expression understanding pattern defined by the SQL standard.

Matching rules:

- a. Similar to **LIKE**, this operator succeeds only when its pattern matches the entire string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign.
- b. The underscore (**_**) represents (matching) any single character. Percentage (**%**) indicates the wildcard character of any string.
- c. **SIMILAR TO** supports these pattern-matching metacharacters borrowed from POSIX-style regular expressions:

Metacharacter	Description
	Specifies alternation (either of two alternatives).
*	Specifies repetition of the previous item zero or more times.

Metacharacter	Description
+	Specifies repetition of the previous item one or more times.
?	Specifies repetition of the previous item zero or one time.
{m}	Specifies repetition of the previous item exactly <i>m</i> times.
{m,}	Specifies repetition of the previous item <i>m</i> or more times.
{m,n}	Specifies repetition of the previous item at least <i>m</i> times and does not exceed <i>n</i> times.
()	Specifies that parentheses () can be used to group items into a single logical item.
[...]	Specifies a character class, just as in POSIX-style regular expressions.

- d. A preamble escape character disables the special meaning of any of these metacharacters. The rules for using escape characters are the same as those for LIKE.

Regular expressions:

The **substring(string from pattern for escape)** function extracts a substring that matches an SQL regular expression pattern.

Example:

```
gaussdb=# SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO 'a' AS RESULT;
result
-----
f
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO '(b|c)%' AS RESULT;
result
-----
f
(1 row)
```

- POSIX-style regular expressions

Description: A regular expression is a collation that is an abbreviated definition of a set of strings (a regular set). If a string is a member of a regular expression described by a regular expression, the string matches the regular expression. POSIX-style regular expressions provide a more powerful

means for pattern matching than the LIKE and SIMILAR TO operators. [Table 7-29](#) lists all available operators for pattern matching using POSIX-style regular expressions.

Table 7-29 Regular expression match operators

Operator	Description	Example
~	Matches a regular expression, which is case-sensitive.	'thomas' ~ '.*thomas.*'
~*	Matches a regular expression, which is case-insensitive.	'thomas' ~* '.*Thomas.*'
!~	Does not match a regular expression, which is case-sensitive.	'thomas' !~ '.*Thomas.*'
!~*	Does not match a regular expression, which is case-insensitive.	'thomas' !~* '.*vadim.*'

Matching rules:

- Unlike LIKE patterns, a regular expression is allowed to match anywhere within a string, unless the regular expression is explicitly anchored to the beginning or end of the string.
- Besides the metacharacters mentioned above, POSIX-style regular expressions also support the following pattern matching metacharacters:

Metacharacter	Description
^	Specifies the match starting with a string.
\$	Specifies the match at the end of a string.
.	Matches any single character.

Regular expressions:

POSIX-style regular expressions support the following functions:

- The [substring\(string from pattern\)](#) function provides a method for extracting a substring that matches the POSIX-style regular expression pattern.
- The [regexp_count\(string text, pattern text \[, position int \[, flags text\]\]\)](#) function provides the function of obtaining the number of substrings that match the POSIX-style regular expression pattern.

- The **regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]])** function is used to obtain the position of a substring that matches a POSIX-style regular expression pattern.
- The **regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])** function provides a method to extract a substring that matches a POSIX-style regular expression pattern.
- The **regexp_replace(string, pattern, replacement [,flags])** function replaces the substring that matches the POSIX-style regular expression pattern with the new text.
- The **regexp_matches(string text, pattern text [, flags text])** function returns a text array consisting of all captured substrings that match a POSIX-style regular expression pattern.
- The **regexp_split_to_table(string text, pattern text [, flags text])** function splits a string using a POSIX-style regular expression pattern as a delimiter.
- The **regexp_split_to_array(string text, pattern text [, flags text])** function behaves the same as `regexp_split_to_table`, except that `regexp_split_to_array` returns its result as an array of text.

NOTE

The regular expression split functions ignore zero-length matches, which occur at the beginning or end of a string or after the previous match. This is contrary to the strict definition of regular expression matching. The latter is implemented by **regexp_matches**, but the former is usually the most commonly used behavior in practice.

Example:

```
gaussdb=# SELECT 'abc' ~ 'Abc' AS RESULT;
result
-----
f
(1 row)
gaussdb=# SELECT 'abc' ~* 'Abc' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' !~ 'Abc' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' !~* 'Abc' AS RESULT;
result
-----
f
(1 row)
gaussdb=# SELECT 'abc' ~ '^a' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' ~ '(b|d)' AS RESULT;
result
-----
t
(1 row)
gaussdb=# SELECT 'abc' ~ '^ (b|c)' AS RESULT;
result
```

```
-----  
f  
(1 row)
```

Although most regular expression searches can be executed quickly, they can still be artificially processed to require any length of time and any amount of memory. It is not recommended that you accept the regular expression search pattern from the non-security pattern source. If you must do this, you are advised to add the statement timeout limit. The search with the SIMILAR TO pattern has the same security risks as the SIMILAR TO provides many capabilities that are the same as those of the POSIX-style regular expression. The LIKE search is much simpler than the other two options. Therefore, it is more secure to accept the non-secure pattern source search.

7.5.7 Arithmetic Functions and Operators

Arithmetic Operators

- +

Description: Addition

Example:

```
gaussdb=# SELECT 2+3 AS RESULT;  
result  
-----  
5  
(1 row)
```

- -

Description: Subtraction

Example:

```
gaussdb=# SELECT 2-3 AS RESULT;  
result  
-----  
-1  
(1 row)
```

- *

Description: Multiplication

Example:

```
gaussdb=# SELECT 2*3 AS RESULT;  
result  
-----  
6  
(1 row)
```

- /

Description: Division (The result is not rounded.)

Example:

```
gaussdb=# SELECT 4/2 AS RESULT;  
result  
-----  
2  
(1 row)  
gaussdb=# SELECT 4/3 AS RESULT;  
result  
-----  
1.3333333333333333  
(1 row)
```

- +/-
Description: Positive/Negative
Example:

```
gaussdb=# SELECT -2 AS RESULT;  
result  
-----  
-2  
(1 row)
```
- %
Description: Model (to obtain the remainder)
Example:

```
gaussdb=# SELECT 5%4 AS RESULT;  
result  
-----  
1  
(1 row)
```
- @
Description: Absolute value
Example:

```
gaussdb=# SELECT @ -5.0 AS RESULT;  
result  
-----  
5.0  
(1 row)
```
- ^
Description: Power (exponent calculation)
Example:

```
gaussdb=# SELECT 2.0^3.0 AS RESULT;  
result  
-----  
8.0000000000000000  
(1 row)
```
- |/
Description: Square root
Example:

```
gaussdb=# SELECT |/ 25.0 AS RESULT;  
result  
-----  
5  
(1 row)
```
- ||/
Description: Cubic root
Example:

```
gaussdb=# SELECT ||/ 27.0 AS RESULT;  
result  
-----  
3  
(1 row)
```
- !
Description: Factorial
Example:

```
gaussdb=# SELECT 5! AS RESULT;  
result
```


Example:

```
gaussdb=# SELECT 8>>2 AS RESULT;  
result  
-----  
      2  
(1 row)
```

Arithmetic Functions

- **abs(x)**
Description: Absolute value
Return type: same as the input

Example:

```
gaussdb=# SELECT abs(-17.4);  
abs  
-----  
 17.4  
(1 row)
```

- **acos(x)**
Description: Arc cosine
Return type: double precision

Example:

```
gaussdb=# SELECT acos(-1);  
acos  
-----  
3.14159265358979  
(1 row)
```

- **asin(x)**
Description: Arc sine
Return type: double precision

Example:

```
gaussdb=# SELECT asin(0.5);  
asin  
-----  
.523598775598299  
(1 row)
```

- **atan(x)**
Description: Arc tangent
Return type: double precision

Example:

```
gaussdb=# SELECT atan(1);  
atan  
-----  
.785398163397448  
(1 row)
```

- **atan2(y, x)**
Description: Arc tangent of y/x
Return type: double precision

Example:

```
gaussdb=# SELECT atan2(2, 1);  
atan2  
-----
```

```
1.10714871779409  
(1 row)
```

- **bitand(integer, integer)**

Description: Performs the AND (&) operation on two integers.

Return type: bigint

Example:

```
gaussdb=# SELECT bitand(127, 63);  
bitand  
-----  
63  
(1 row)
```

- **cbirt(dp)**

Description: Cubic root

Return type: double precision

Example:

```
gaussdb=# SELECT cbirt(27.0);  
cbirt  
-----  
3  
(1 row)
```

- **ceil(x)**

Description: Minimum integer greater than or equal to the parameter

Return type: integer

Example:

```
gaussdb=# SELECT ceil(-42.8);  
ceil  
-----  
-42  
(1 row)
```

- **ceiling(dp or numeric)**

Description: Minimum integer (alias of ceil) greater than or equal to the parameter

Return type: same as the input

Example:

```
gaussdb=# SELECT ceiling(-95.3);  
ceiling  
-----  
-95  
(1 row)
```

- **cos(x)**

Description: Cosine

Return type: double precision

Example:

```
gaussdb=# SELECT cos(-3.1415927);  
cos  
-----  
-.9999999999999999  
(1 row)
```

- **cosh(x)**

Description: Hyperbolic cosine

Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

Example:

```
gaussdb=# SELECT cosh(4);
      cosh
-----
27.3082328360165
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- **cot(x)**

Description: Cotangent

Return type: double precision

Example:

```
gaussdb=# SELECT cot(1);
      cot
-----
.642092615934331
(1 row)
```

- **degrees(dp)**

Description: Converts radians to angles.

Return type: double precision

Example:

```
gaussdb=# SELECT degrees(0.5);
      degrees
-----
28.6478897565412
(1 row)
```

- **div(y numeric, x numeric)**

Description: Integer part of y/x

Return type: numeric

Example:

```
gaussdb=# SELECT div(9,4);
      div
-----
2
(1 row)
```

- **exp(x)**

Description: Natural exponent

Return type: same as the input

Example:

```
gaussdb=# SELECT exp(1.0);
      exp
-----
2.7182818284590452
(1 row)
```

- **floor(x)**

Description: Not larger than the maximum integer of the parameter

Return type: same as the input

Example:

```
gaussdb=# SELECT floor(-42.8);
 floor
-----
   -43
(1 row)
```

- **int1(in)**

Description: Converts the input text parameter to a value of the int1 type and returns the value.

Return type: int1

Example:

```
gaussdb=# SELECT int1('123');
 int1
-----
   123
(1 row)
gaussdb=# SELECT int1('1.1');
 int1
-----
     1
(1 row)
```

 **NOTE**

- When **sql_compatibility** is set to 'MYSQL', non-integer characters are automatically truncated or the value **0** is returned.
 - When **sql_compatibility** is not set to 'MYSQL', an error message is displayed for non-integer characters, indicating that the input is invalid.
- **int2(in)**

Description: Converts the input parameter to a value of the int2 type and returns the value. The supported input parameter types include bigint, float4, float8, int16, integer, numeric, real, and text.

Return type: int2

Example:

```
gaussdb=# SELECT int2('1234');
 int2
-----
   1234
(1 row)
gaussdb=# SELECT int2(25.3);
 int2
-----
     25
(1 row)
```

 **NOTE**

- When **sql_compatibility** is set to 'MYSQL', non-integer characters are automatically truncated or the value **0** is returned.
 - When **sql_compatibility** is not set to 'MYSQL', an error message is displayed for non-integer characters, indicating that the input is invalid.
- **int4(in)**

Description: Converts the input parameter to a value of the int4 type and returns the value. The supported input parameter types include bit, Boolean, char, double precision, int16, numeric, real, smallint, and text

Return type: int4

Example:

```
gaussdb=# SELECT int4('789');
int4
-----
 789
(1 row)
gaussdb=# SELECT int4(99.9);
int4
-----
 100
(1 row)
```

 NOTE

- When **sql_compatibility** is set to 'MYSQL', non-integer characters are automatically truncated or the value **0** is returned.
 - When **sql_compatibility** is not set to 'MYSQL', an error message is displayed for non-integer characters, indicating that the input is invalid.
- **int8(in)**

Description: Converts the input parameter to a value of the int8 type and returns the value. The supported input parameter types include bit, double precision, int16, integer, numeric, oid, real, smallint, and text.

Return type: int8

Example:

```
gaussdb=# SELECT int8('789');
int8
-----
 789
(1 row)
gaussdb=# SELECT int8(99.9);
int8
-----
  99
(1 row)
```

 NOTE

- When **sql_compatibility** is set to 'MYSQL', non-integer characters are automatically truncated or the value **0** is returned.
 - When **sql_compatibility** is not set to 'MYSQL', an error message is displayed for non-integer characters, indicating that the input is invalid.
- **float4(in)**

Description: Converts the input parameter to a value of the float4 type and returns the value. The supported input parameter types include bigint, double precision, int16, integer, numeric, smallint, and text.

Return type: float4

Example:

```
gaussdb=# SELECT float4('789');
float4
-----
 789
(1 row)
gaussdb=# SELECT float4(99.9);
float4
-----
 99.9
(1 row)
```

- float8(in)

Description: Converts the input parameter to a value of the float8 type and returns the value. The supported input parameter types include bigint, int16, integer, numeric, real, smallint, and text.

Return type: float8

Example:

```
gaussdb=# SELECT float8('789');
 float8
-----
    789
(1 row)
```

```
gaussdb=# SELECT float8(99.9);
 float8
-----
    99.9
(1 row)
```

- int16(in)

Description: Converts the input parameter to a value of the int16 type and returns the value. The supported input parameter types include bigint, Boolean, double precision, integer, numeric, oid, real, smallint, and tinyint.

Return type: int16

Example:

```
gaussdb=# SELECT int16('789');
 int16
-----
    789
(1 row)
```

```
gaussdb=# SELECT int16(99.9);
 int16
-----
    100
(1 row)
```

- numeric(in)

Description: Converts the input parameter to a value of the numeric type and returns the value. The supported input parameter types include bigint, Boolean, double precision, int16, integer, money, real, and smallint.

Return type: numeric

Example:

```
gaussdb=# SELECT "numeric"('789');
 numeric
-----
    789
(1 row)
```

```
gaussdb=# SELECT "numeric"(99.9);
 numeric
-----
    99.9
(1 row)
```

- oid(in)

Description: Converts the input parameter to a value of the oid type and returns the value. The supported input parameter types include bigint and int16.

Return type: oid

- **radians(dp)**
Description: Converts angles to radians.
Return type: double precision
Example:

```
gaussdb=# SELECT radians(45.0);
radians
-----
.785398163397448
(1 row)
```
- **random()**
Description: Random number between 0.0 and 1.0
Return type: double precision
Example:

```
gaussdb=# SELECT random();
random
-----
.824823560658842
(1 row)
```
- **multiply(x double precision or text, y double precision or text)**
Description: product of x and y.
Return type: double precision
Example:

```
gaussdb=# SELECT multiply(9.0, '3.0');
multiply
-----
27
(1 row)
gaussdb=# SELECT multiply('9.0', 3.0);
multiply
-----
27
(1 row)
```
- **ln(x)**
Description: Natural logarithm
Return type: same as the input
Example:

```
gaussdb=# SELECT ln(2.0);
ln
-----
.6931471805599453
(1 row)
```
- **log(x)**
Description: Logarithm with 10 as the base
Return type: same as the input
Example:

```
gaussdb=# SELECT log(100.0);
log
-----
2.0000000000000000
(1 row)
```
- **log(b numeric, x numeric)**
Description: Logarithm with b as the base

Return type: numeric

Example:

```
gaussdb=# SELECT log(2.0, 64.0);
log
-----
6.000000000000000000
(1 row)
```

- **mod(x,y)**

Description:

Remainder of x/y (model)

If x equals to 0, y is returned.

Return type: same as the parameter type

Example:

```
gaussdb=# SELECT mod(9,4);
mod
-----
1
(1 row)
gaussdb=# SELECT mod(9,0);
mod
-----
9
(1 row)
```

- **pi()**

Description: π constant value

Return type: double precision

Example:

```
gaussdb=# SELECT pi();
pi
-----
3.14159265358979
(1 row)
```

- **power(a double precision or numeric, b double precision or numeric)**

Description: b power of a

Return type: double precision or numeric. If implicit type conversion is not considered and both input parameters are of the numeric type, the numeric type is returned. If either input parameter is of the double precision type, the double precision type is returned.

Example:

```
gaussdb=# SELECT power(9.0, 3.0);
power
-----
729.0000000000000000
(1 row)
```

- **remainder(x,y)**

Description: Remainder of x/y. If y is 0, an error is reported.

Return type: same as the input (float4, float8, or numeric)

Example:

```
gaussdb=# SELECT remainder(11,4);
remainder
-----
-1
(1 row)
```

```
gaussdb=# SELECT remainder(9,0);  
ERROR: division by zero  
CONTEXT: referenced column: remainder
```

NOTE

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- **round(x)**

Description: Integer closest to the input parameter

Return type: same as the input (double precision or numeric)

Example:

```
gaussdb=# SELECT round(42.4);  
round  
-----  
42  
(1 row)  
  
gaussdb=# SELECT round(42.6);  
round  
-----  
43  
(1 row)
```

CAUTION

The output of the float/double type may be **-0**. (This also occurs in functions such as **trunc** and **ceil**. If the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s1** in an ORA-compatible database, the returned result is **0**. The following is an example:

```
gaussdb=# SELECT round(-0.2::float8);  
round  
-----  
-0  
(1 row)
```

- **round(v numeric, s int)**

Description: **s** digits are kept after the decimal point.

Return type: numeric

Example:

```
gaussdb=# SELECT round(42.4382, 2);  
round  
-----  
42.44  
(1 row)
```

NOTE

When the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s1** in an ORA-compatible database, **s** is truncated to an integer. Otherwise, **s** is rounded off to an integer.

When the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1** in an ORA-compatible database, the **round** function supports **round(timestamp, text)** overloading. When **(text, text)** or **(text, '')** is used as the input parameter to call the **round** function, **round(timestamp, text)** is preferred.

- **setseed(dp)**

Description: Sets seed for the following random() calling (between -1.0 and 1.0, inclusive).

Return type: void

Example:

```
gaussdb=# SELECT setseed(0.54823);
 setseed
-----
(1 row)
```

- sign(x)

Description: Returns symbols of this parameter.

Return type: **-1** indicates negative numbers. **0** indicates 0, and **1** indicates positive numbers.

Example:

```
gaussdb=# SELECT sign(-8.4);
 sign
-----
 -1
(1 row)
```

- sin(x)

Description: Sine

Return type: double precision

Example:

```
gaussdb=# SELECT sin(1.57079);
 sin
-----
 .999999999979986
(1 row)
```

- sinh(x)

Description: Hyperbolic sine

Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

Example:

```
gaussdb=# SELECT sinh(4);
 sinh
-----
 27.2899171971277
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- sqrt(x)

Description: Square root

Return type: same as the input

Example:

```
gaussdb=# SELECT sqrt(2.0);
 sqrt
-----
 1.414213562373095
(1 row)
```

- **tan(x)**

Description: Tangent

Return type: double precision

Example:

```
gaussdb=# SELECT tan(20);
tan
-----
2.23716094422474
(1 row)
```

- **tanh(x)**

Description: Hyperbolic tangent

Return type: same as the input (double precision or numeric)

Example:

```
gaussdb=# SELECT tanh(0.1);
tanh
-----
0.0996679946249558171183050836783521835389
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

- **trunc(x)**

Description: Truncates (the integral part).

Return type: same as the input

Example:

```
gaussdb=# SELECT trunc(42.8);
trunc
-----
42
(1 row)
```

- **trunc(v numeric, s int)**

Description: Truncates a number with **s** digits after the decimal point.

Return type: numeric

Example:

```
gaussdb=# SELECT trunc(42.4382, 2);
trunc
-----
42.43
(1 row)
```

 **NOTE**

In an ORA-compatible database, this function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is a valid value under the compatible configuration item. If the value of **s** is a decimal, the value is truncated instead of being rounded off.

- **width_bucket(op numeric, b1 numeric, b2 numeric, count int)**

Description: Returns a bucket to which the operand will be assigned in an equi-depth histogram with **count** buckets, ranging from **b1** to **b2**.

Return type: int

Example:

```
gaussdb=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
width_bucket
-----
          3
(1 row)
```

- **width_bucket(op dp, b1 dp, b2 dp, count int)**
Description: Returns a bucket to which the operand will be assigned in an equi-depth histogram with **count** buckets, ranging from **b1** to **b2**.
Return type: int
Example:

```
gaussdb=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
width_bucket
-----
          3
(1 row)
```
- **smgrne(a smgr, b smgr)**
Description: Compares two integers of the smgr type to check whether they are different.
Return type: Boolean
- **smgreq(a smgr, b smgr)**
Description: Compares two integers of the smgr type to check whether they are equivalent.
Return type: Boolean
- **int1abs**
Description: Returns the absolute value of data of the uint8 type.
Parameter: tinyint
Return type: tinyint
- **int1and**
Description: Returns the bitwise AND result of two data records of the uint8 type.
Parameter: tinyint, tinyint
Return type: tinyint
- **int1cmp**
Description: Returns the comparison result of two data records of the uint8 type. If the value of the first parameter is greater, **1** is returned. If the value of the second parameter is greater, **-1** is returned. If they are the same, **0** is returned.
Parameter: tinyint, tinyint
Return type: integer
- **int1div**
Description: Returns the result of dividing two data records of the uint8 type. The result is of the float8 type.
Parameter: tinyint, tinyint
Return type: tinyint
- **int1eq**
Description: Compares two pieces of data of the uint8 type to check whether they are the same.

Parameter: tinyint, tinyint

Return type: Boolean

- int1ge

Description: Determines whether the value of the first parameter is greater than or equal to the value of the second parameter in two data records of the uint8 type.

Parameter: tinyint, tinyint

Return type: Boolean

- int1gt

Description: Performs the greater-than operation on an unsigned 1-byte integer.

Parameter: tinyint, tinyint

Return type: Boolean

- int1larger

Description: Returns the larger value of unsigned one-byte integers.

Parameter: tinyint, tinyint

Return type: tinyint

- int1le

Description: Determines whether the unsigned 1-byte integer is less than or equal to.

Parameter: tinyint, tinyint

Return type: Boolean

- int1lt

Description: Determines whether the unsigned 1-byte integer is less than.

Parameter: tinyint, tinyint

Return type: Boolean

- int1smaller

Description: Returns the smaller of two unsigned one-byte integers.

Parameter: tinyint, tinyint

Return type: tinyint

- int1inc

Description: Performs an addition operation on an unsigned 1-byte integer.

Parameter: tinyint

Return type: tinyint

- int1mi

Description: Performs a minus operation on an unsigned 1-byte integer.

Parameter: tinyint, tinyint

Return type: tinyint

- int1mod

Description: Performs a remainder operation on an unsigned 1-byte integer.

Parameter: tinyint, tinyint

Return type: tinyint

- **int1mul**
Description: Performs a multiplication operation on an unsigned 1-byte integer.
Parameter: tinyint, tinyint
Return type: tinyint
- **int1ne**
Description: Performs a not-equal-to operation on an unsigned 1-byte integer.
Parameter: tinyint, tinyint
Return type: Boolean
- **int1pl**
Description: Performs an addition operation on an unsigned 1-byte integer.
Parameter: tinyint, tinyint
Return type: tinyint
- **int1um**
Description: Returns an unsigned 2-byte integer after subtracting the opposite number from the unsigned 1-byte integer.
Parameter: tinyint
Return type: smallint
- **int1xor**
Description: Performs an exclusive OR operation on an unsigned 1-byte integer.
Parameter: tinyint, tinyint
Return type: tinyint
- **cash_div_int1**
Description: Performs a division operation on the money type.
Parameter: money, tinyint
Return type: money
- **cash_mul_int1**
Description: Performs a multiplication operation on the money type.
Parameter: money, tinyint
Return type: money
- **int1not**
Description: Reverts binary bits of an unsigned 1-byte integer.
Parameter: tinyint
Return type: tinyint
- **int1or**
Description: Performs an OR operation on an unsigned 1-byte integer.
Parameter: tinyint, tinyint
Return type: tinyint
- **int1shl**
Description: Shifts an unsigned 1-byte integer leftwards by a specified number of bits.

Parameter: tinyint, integer

Return type: tinyint

- int1shr

Description: Shifts an unsigned 1-byte integer rightwards by a specified number of bits.

Parameter: tinyint, integer

Return type: tinyint

- analyze_tgtype_for_type(n smallint)

Description: Parses **pg_trigger.tgtype**, parses *n* by bit, and returns one of **before each row**, **after each row**, **before statement**, **after statement**, and **instead of**.

Return type: varchar2(16)

- analyze_tgtype_for_event(n smallint)

Description: Parses **pg_trigger.tgtype**, parses *n* by bit, and returns one or more of **insert**, **update**, **delete**, and **truncate**.

Return type: varchar2(246)

- nanvl(n2, n1)

Description: Two parameters are entered. The parameters must be of the numeric type or a non-numeric type that can be implicitly converted to the numeric type. If the first parameter **n2** is NaN, **n1** is returned. Otherwise, **n2** is returned.

Return value type: input parameter with a higher priority. The priority is as follows: double precision > float4 > numeric.

Example:

```
gaussdb=# SELECT nanvl('NaN', 1.1);
nanvl
-----
1.1
(1 row)
```

NOTE

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s2** in an ORA-compatible database.

7.5.8 Date and Time Processing Functions and Operators

Date and Time Operators

For details about the time and date operators, see [Table 7-30](#).

 **NOTE**

Do not use expressions similar to 'now'::date, 'now'::timestamp,'now'::timestampz (for string constant conversion) and text_date('now') to obtain the current database time or use the current time as the input parameter of the function. In these scenarios, the optimizer calculates the constant time in advance, causing incorrect query results.

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b='now'::date;
QUERY PLAN
-----
Seq Scan on t1 (cost=0.00..13.60 rows=1 width=310)
Filter: ((b)::text = '2024-11-09 15:07:56'::text)
(2 rows)
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=text_date('now');
QUERY PLAN
-----
Seq Scan on t1 (cost=0.00..13.60 rows=1 width=310)
Filter: ((b)::text = '2024-11-09'::text)
(2 rows)
```

You are advised to use the now() and currenttimestamp() functions to obtain the current time of the database.

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=now();
QUERY PLAN
-----
Seq Scan on t1 (cost=0.00..14.80 rows=1 width=310)
Filter: ((b)::text = (now())::text)
(2 rows)
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=text_date(now());
QUERY PLAN
-----
Seq Scan on t1 (cost=0.00..16.00 rows=1 width=310)
Filter: ((b)::text = (text_date((now())::text))::text)
(2 rows)
```

When the user uses date and time operators, explicit type prefixes are modified for corresponding operands to ensure that the operands parsed by the database are consistent with what the user expects, and no unexpected results occur.

For example, abnormal mistakes will occur in the following example without an explicit data type.

```
SELECT date '2001-10-01' - '7' AS RESULT;
```

Table 7-30 Time and date operators

Operator	Example
+	<pre>gaussdb=# SELECT date '2001-9-28' + integer '7' AS RESULT; result ----- 2001-10-05 (1 row)</pre> <p>NOTE In ORA-compatible mode, the query result is 2001-10-05 00:00:00.</p>
	<pre>gaussdb=# SELECT date '2001-09-28' + interval '1 hour' AS RESULT; result ----- 2001-09-28 01:00:00 (1 row)</pre>

Operator	Example
	<pre>gaussdb=# SELECT date '2001-09-28' + time '03:00' AS RESULT; result ----- 2001-09-28 03:00:00 (1 row)</pre> <pre>gaussdb=# SELECT interval '1 day' + interval '1 hour' AS RESULT; result ----- 1 day 01:00:00 (1 row)</pre> <pre>gaussdb=# SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT; result ----- 2001-09-29 00:00:00 (1 row)</pre> <pre>gaussdb=# SELECT time '01:00' + interval '3 hours' AS RESULT; result ----- 04:00:00 (1 row)</pre>
-	<pre>gaussdb=# SELECT date '2001-10-01' - date '2001-09-28' AS RESULT; result ----- 3 (1 row)</pre> <pre>gaussdb=# SELECT date '2001-10-01' - integer '7' AS RESULT; result ----- 2001-09-24 00:00:00 (1 row)</pre> <pre>gaussdb=# SELECT date '2001-09-28' - interval '1 hour' AS RESULT; result ----- 2001-09-27 23:00:00 (1 row)</pre> <pre>gaussdb=# SELECT time '05:00' - time '03:00' AS RESULT; result ----- 02:00:00 (1 row)</pre> <pre>gaussdb=# SELECT time '05:00' - interval '2 hours' AS RESULT; result ----- 03:00:00 (1 row)</pre> <pre>gaussdb=# SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT; result ----- 2001-09-28 00:00:00 (1 row)</pre> <pre>gaussdb=# SELECT interval '1 day' - interval '1 hour' AS RESULT; result ----- 23:00:00 (1 row)</pre>

Operator	Example
	<pre>gaussdb=# SELECT timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' AS RESULT; result ----- 1 day 15:00:00 (1 row)</pre>
*	<pre>gaussdb=# SELECT 900 * interval '1 second' AS RESULT; result ----- 00:15:00 (1 row)</pre>
	<pre>gaussdb=# SELECT 21 * interval '1 day' AS RESULT; result ----- 21 days (1 row)</pre>
	<pre>gaussdb=# SELECT double precision '3.5' * interval '1 hour' AS RESULT; result ----- 03:30:00 (1 row)</pre>
/	<pre>gaussdb=# SELECT interval '1 hour' / double precision '1.5' AS RESULT; result ----- 00:40:00 (1 row)</pre>

Time and Date Functions

- age(timestamp, timestamp)**
 Description: Subtracts parameters, producing a result in YYYY-MM-DD format. If the result is negative, the returned result is also negative. The input parameters can contain timezone or not.
 Return type: interval
 Example:

```
gaussdb=# SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
age
-----
43 years 9 mons 27 days
(1 row)
```
- age(timestamp)**
 Description: Subtracts the parameter value from the system time when the current SQL statement starts to be executed. The input parameter may or may not contain a time zone.
 Return type: interval
 Example:

```
gaussdb=# SELECT age(timestamp '1957-06-13');
age
-----
60 years 2 mons 18 days
(1 row)
```

- `clock_timestamp()`

Description: Returns the timestamp of the system time when the current function is called. The volatile function obtains the latest timestamp for each scan. Therefore, the result of each call in a query is different.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT clock_timestamp();
          clock_timestamp
-----
2017-09-01 16:57:36.636205+08
(1 row)
```

- `current_date`

Description: Returns the system date when the current SQL statement starts.

Return type: date

Example:

```
gaussdb=# SELECT current_date;
          date
-----
2017-09-01
(1 row)
```

 **NOTE**

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2** in an ORA-compatible database, the return value type is timestamp.

- `current_time`

Description: Specifies the system time when the current transaction starts.

Return type: time with time zone

Example:

```
gaussdb=# SELECT current_time;
          timetz
-----
16:58:07.086215+08
(1 row)
```

- `current_timestamp`

Description: Returns the system time when the current SQL execution starts. This is a statement-level timestamp. The returned results within the same statement remain unchanged.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT current_timestamp;
          pg_systimestamp
-----
2017-09-01 16:58:19.22173+08
(1 row)
```

- `current_timestamp(precision)`

Description: Returns the system time when the current transaction is started, and rounds the microseconds of the result to the specified decimal places.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT current_timestamp(1);
          timestamptz
```

```
-----
2017-09-01 16:58:19.2+08
(1 row)
```

 **NOTE**

- When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2** in an ORA-compatible database, the **precision** parameter can be an integer of the numeric type. Otherwise, only the int type is supported.
- Zeros at the end of microseconds are not displayed. For example, 2017-09-01 10:32:19.212000 is displayed as 2017-09-01 10:32:19.212.

- **pg_systimestamp()**

Description: Current date and time (start of the current statement).

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT pg_systimestamp();
           pg_systimestamp
```

```
-----
2015-10-14 11:21:28.317367+08
(1 row)
```

- **date_part(text, timestamp)**

Description:

Retrieves fields such as year or hour from date/time values.

It is equivalent to **extract(field from timestamp)**.

Timestamp types: abstime, date, interval, reltime, time with time zone, time without time zone, timestamp with time zone, timestamp without time zone

Return type: double precision

Example:

```
gaussdb=# SELECT date_part('hour', timestamp '2001-02-16 20:38:40');
           date_part
```

```
-----
                20
(1 row)
```

- **date_part(text, interval)**

Description: Obtains the month. If the value is greater than 12, obtain the remainder after it is divided by 12. It is equivalent to **extract(field from timestamp)**.

Return type: double precision

Example:

```
gaussdb=# SELECT date_part('month', interval '2 years 3 months');
           date_part
```

```
-----
                3
(1 row)
```

- **date_trunc(text, timestamp)**

Description: Truncates to the precision specified by **text**.

Return type: interval, timestamp with time zone, timestamp without time zone

Example:

```
gaussdb=# SELECT date_trunc('hour', timestamp '2001-02-16 20:38:40');
           date_trunc
```

```
2001-02-16 20:00:00
(1 row)
```

- **trunc(timestamp)**

Description: Truncates to day by default.

Example:

```
gaussdb=# SELECT trunc(timestamp '2001-02-16
20:38:40');
          trunc
-----
2001-02-16 00:00:00
(1 row)
```

- **trunc(arg1, arg2)**

Description: Truncates to the precision specified by **arg2**.

- Type of **arg1**: interval, timestamp with time zone, timestamp without time zone
- Type of **arg2**: text

Return type: interval, timestamp with time zone, timestamp without time zone

Example:

```
gaussdb=# SELECT trunc(timestamp '2001-02-16 20:38:40',
'hour');
          trunc
-----
2001-02-16 20:00:00
(1 row)
```

- **round(arg1, arg2)**

Description: Rounds off to the precision specified by **arg2**.

Type of **arg1**: timestamp without time zone

Type of **arg2**: text

Return type: timestamp without time zone

Example:

```
gaussdb=# SELECT round(timestamp '2001-02-16 20:38:40',
'hour');
          round
-----
2001-02-16 21:00:00
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s1** in an ORA-compatible database.

- **daterange(arg1, arg2)**

Description: Obtains time boundary information.

arg1 type: date

arg2 type: date

Return type: daterange

Example:

```
gaussdb=# select daterange('2000-05-06','2000-08-08');
          daterange
-----
[2000-05-06,2000-08-08)
(1 row)
```

- daterange(arg1, arg2, text)**
 Description: Obtains time boundary information.
arg1 type: date
arg2 type: date
text type: text
 Return type: daterange
 Example:

```
gaussdb=# select daterange('2000-05-06','2000-08-08','[]');
          daterange
-----
 [2000-05-06,2000-08-09)
 (1 row)
```
- extract(field from timestamp)**
 Description: Obtains the hour.
 Return type: double precision
 Example:

```
gaussdb=# SELECT extract(hour from timestamp '2001-02-16 20:38:40');
          date_part
-----
             20
 (1 row)
```
- extract(field from interval)**
 Description: Obtains the month. If the value is greater than 12, obtain the remainder after it is divided by 12.
 Return type: double precision
 Example:

```
gaussdb=# SELECT extract(month from interval '2 years 3 months');
          date_part
-----
             3
 (1 row)
```
- isfinite(date)**
 Description: Checks whether a date is a finite value. If the date is a finite value, **t** is returned. Otherwise, **f** is returned.
 Return type: Boolean
 Example:

```
gaussdb=# SELECT isfinite(date '2001-02-16');
          isfinite
-----
             t
 (1 row)
gaussdb=# SELECT isfinite(date 'infinity');
          isfinite
-----
             f
 (1 row)
```
- isfinite(timestamp)**
 Description: Checks whether a timestamp is a finite value. If the timestamp is a finite value, **t** is returned. Otherwise, **f** is returned.
 Return type: Boolean
 Example:

```
gaussdb=# SELECT isfinite(timestamp '2001-02-16 21:28:30');
isfinite
-----
t
(1 row)
gaussdb=# SELECT isfinite(timestamp 'infinity');
isfinite
-----
f
(1 row)
```

- **isfinite(interval)**

Description: Checks whether the interval is a finite value. If the interval is a finite value, **t** is returned. Currently, **f** cannot be returned. If '**infinity**' is entered, an error is reported.

Return type: Boolean

Example:

```
gaussdb=# SELECT isfinite(interval '4 hours');
isfinite
-----
t
(1 row)
```

- **justify_days(interval)**

Description: Adjusts intervals to 30-day time periods, which are represented as months.

Return type: interval

Example:

```
gaussdb=# SELECT justify_days(interval '35 days');
justify_days
-----
1 mon 5 days
(1 row)
```

- **justify_hours(interval)**

Description: Sets the time interval in days (24 hours is one day).

Return type: interval

Example:

```
gaussdb=# SELECT JUSTIFY_HOURS(INTERVAL '27 HOURS');
justify_hours
-----
1 day 03:00:00
(1 row)
```

- **justify_interval(interval)**

Description: Adjusts **interval** using **justify_days** and **justify_hours**.

Return type: interval

Example:

```
gaussdb=# SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
justify_interval
-----
29 days 23:00:00
(1 row)
```

- **localtime**

Description: Specifies the system time when the current transaction starts.

Return type: time

Example:

```
gaussdb=# SELECT localtime AS RESULT;
      result
-----
16:05:55.664681
(1 row)
```

- localtimestamp

Description: Returns the system date and time when the current SQL query execution starts.

Return type: timestamp

Example:

```
gaussdb=# SELECT localtimestamp;
      timestamp
-----
2017-09-01 17:03:30.781902
(1 row)
```

- now()

Description: Returns the system date and time when the current transaction starts. The results returned in the same transaction are the same.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT now();
      now
-----
2017-09-01 17:03:42.549426+08
(1 row)
```

- timenow()

Description: Returns the system date and time when the current SQL query execution starts.

Return type: abstime

Example:

```
gaussdb=# select timenow();
      timenow
-----
2020-06-23 20:36:56+08
(1 row)
```

- numtodsinterval(num, interval_unit)

Description: Converts a number to the interval type. **num** is a numeric-typed number. **interval_unit** is a string in the following format: 'DAY' | 'HOUR' | 'MINUTE' | 'SECOND'

You can set the GUC parameter **IntervalStyle** to **ORA** to be compatible with the interval output format of the function in the Oracle Database.

Return type: interval

Example:

```
gaussdb=# SELECT numtodsinterval(100, 'HOUR');
      numtodsinterval
-----
100:00:00
(1 row)

gaussdb=# SET intervalstyle = oracle;
SET
gaussdb=# SELECT numtodsinterval(100, 'HOUR');
      numtodsinterval
-----
```

```
+000000004 04:00:00.000000000
(1 row)
```

 **NOTE**

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2** in an ORA-compatible database, an error is reported if **interval_unit** is set to **'DAY'** and **num** is set to a value greater than **1000000000**.

- `numtoyminterval(num, interval_unit)`

Description: Converts a number to the interval type. **num** is a number of the numeric type, and **interval_unit** is a string of the fixed format ('YEAR'|'MONTH').

You can set the GUC parameter **IntervalStyle** to **ORA** to be compatible with the interval output format in the Oracle Database.

Return type: interval

Example:

```
gaussdb=# \c gaussdb_o;
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "gaussdb_o" as user "omm".
gaussdb_o=# set a_format_version='10c';
SET
gaussdb_o=# set a_format_dev_version='s2';
SET
gaussdb=# SELECT numtoyminterval(100, 'MONTH');
 numtoyminterval
-----
8 years 4 mons
(1 row)
gaussdb_o=# SET intervalstyle = oracle;
SET
gaussdb_o=# SELECT numtoyminterval(100, 'MONTH');
 numtoyminterval
-----
8-4
(1 row)

gaussdb_o=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "omm".
gaussdb=# DROP DATABASE gaussdb_o;
DROP DATABASE
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an ORA-compatible database.

- `new_time(date, timezone1,timezone2)`

Description: Returns the date and time of the time zone specified by **timezone2** when the date and time of the time zone specified by **timezone1** are date.

Return type: timestamp

Example:

```
gaussdb=# select new_time('1997-10-10','AST','EST');
 new_time
-----
1997-10-09 23:00:00
(1 row)
gaussdb=# SELECT NEW_TIME(TO_TIMESTAMP ('10-Sep-02 14:10:10.123000','DD-Mon-RR
HH24:MI:SS.FF'), 'AST', 'PST');
 new_time
```

```
-----
2002-09-10 10:10:10.123
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an ORA-compatible database.

- **sessiontimezone**

Description: Returns the time zone of the current session. There is no input parameter.

Return type: text

Example:

```
gaussdb=# SELECT SESSIONTIMEZONE;
session_time_zone
-----
PST8PDT
(1 row)
gaussdb=# SELECT LOWER(SESSIONTIMEZONE);
lower
-----
@ 8 hours
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an ORA-compatible database.

If the value of **session time zone** is **GMT+08:00** or **GMT-08:00**, the offset of the positive value is used as the position west of Greenwich Mean Time (GMT). For example, **GMT+08:00** indicates GMT-08:00 and **GMT-08:00** indicates GMT+08:00.

- **sys_extract_utc(timestamp| timestamptz)**

Description: Extracts Coordinated Universal Time (UTC, also formerly known as Greenwich Mean Time) from a date-time value with a time zone offset or time zone region name. If no time zone is specified, the date and time are associated with the session time zone. The input parameter can be in timestamp or timestamp format.

Return type: timestamp

Example:

```
gaussdb=# SELECT SYS_EXTRACT_UTC(TIMESTAMP '2000-03-28 11:30:00.00');
sys_extract_utc
-----
2000-03-28 03:30:00
(1 row)
gaussdb=# SELECT SYS_EXTRACT_UTC(TIMESTAMPtz '2000-03-28 11:30:00.00 -08:00');
sys_extract_utc
-----
2000-03-28 19:30:00
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an ORA-compatible database.

- **tz_offset('time_zone_name' | '(+/-)hh:mi' | SESSIONTIMEZONE | DBTIMEZONE)**

Description: Returns the UTC offset of the time zone indicated by the input parameter. The input parameter has the preceding four formats.

Return type: text

Example:

```
gaussdb=# SELECT TZ_OFFSET('US/Pacific');
tz_offset
-----
-08:00
(1 row)
gaussdb=# SELECT TZ_OFFSET(sessiontimezone);
tz_offset
-----
+08:00
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an ORA-compatible database.

- **pg_sleep(seconds)**

Description: Specifies the delay time of the server thread in unit of second.

Note that when the database calls this function, the corresponding transaction snapshot is obtained, which is equivalent to a long transaction. If the input parameter time is too long, the database **oldestxmin** may fail to be executed, affecting the table recycling and query performance.

Return type: void

Example:

```
gaussdb=# SELECT pg_sleep(10);
pg_sleep
-----
(1 row)
```

- **statement_timestamp()**

Description: Current date and time (start of the current statement).

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT statement_timestamp();
statement_timestamp
-----
2017-09-01 17:04:39.119267+08
(1 row)
```

- **sysdate**

Description: Returns the system date and time when the current SQL statement is executed.

Return type: timestamp

Example:

```
gaussdb=# SELECT sysdate;
sysdate
-----
2017-09-01 17:04:49
(1 row)
```

- **current_sysdate**

Description: Returns the system date and time when the current SQL query execution starts.

Return type: timestamp

Example:

```
gaussdb=# SELECT current_sysdate();
current_sysdate
-----
2023-06-20 20:09:02
(1 row)
```

- `timeofday()`

Description: Returns the timestamp (such as **clock_timestamp**, but the return type is text) of the system time when the current function is called.

Return type: text

Example:

```
gaussdb=# SELECT totimeofday();
timeofday
-----
Fri Sep 01 17:05:01.167506 2017 CST
(1 row)
```

- `transaction_timestamp()`

Description: Specifies the system date and time when the current transaction starts.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT transaction_timestamp();
transaction_timestamp
-----
2017-09-01 17:05:13.534454+08
(1 row)
```

- `add_months(d,n)`

Description: Returns the date *date* plus *integer* months.

d: indicates the value of the timestamp type and the value that can be implicitly converted to the timestamp type.

n: indicates the value of the INTEGER type and the value that can be implicitly converted to the INTEGER type.

Return type: timestamp

Example:

```
gaussdb=# SELECT add_months(to_date('2017-5-29', 'yyyy-mm-dd'), 11) FROM sys_dummy;
add_months
-----
2018-04-29 00:00:00
(1 row)
```

NOTE

In the scenario where this function is in an ORA-compatible database, the value of **a_format_version** is **10c**, and that of **a_format_dev_version** is **s1**:

- If the calculation result is greater than 9999, an error is reported.
- If the value of *n* is a decimal, the value is truncated instead of being rounded off.

- `last_day(d)`

Description: Returns the date of the last day of the month that contains *date*.

Return type: timestamp

Example:

```
gaussdb=# SELECT last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;
cal_result
-----
```

```
2017-01-31 00:00:00
(1 row)
```

- `months_between(d1, d2)`

Description: Calculates the month difference between time points **d1** and **d2**. If both dates are the end of a month or are the same day, an integer is returned. Otherwise, the return value is a decimal and is calculated as 31 days per month.

Return type: numeric

Example:

```
gaussdb=# SELECT months_between(to_date('2022-10-31', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
 months_between
```

```
-----
          1
(1 row)
```

```
gaussdb=# SELECT months_between(to_date('2022-10-30', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
 months_between
```

```
-----
          1
(1 row)
```

```
gaussdb=# SELECT months_between(to_date('2022-10-29', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
 months_between
```

```
-----
.96774193548387096774
(1 row)
```

 **NOTE**

This function is valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s1** in an ORA-compatible database.

- `next_day(x,y)`

Description: Calculates the time of the next week **y** started from **x**.

Return type: timestamp

Example:

```
gaussdb=# SELECT next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
 cal_result
```

```
-----
2017-05-28 00:00:00
(1 row)
```

 **NOTE**

- The second parameter of **next_day** supports abbreviations (case-insensitive), including **SUN, MON, TUE, WED, THU, THUR, FRI, and SAT**.
- `tinterval(abstime, abstime)`

Description: Creates a time interval with two pieces of absolute time.

Return type: tinterval

Example:

```
gaussdb=# call tinterval(abstime 'May 10, 1947 23:59:12', abstime 'Mon May 1 00:30:30 1995');
 tinterval
```

```
-----
["1947-05-10 23:59:12+08" "1995-05-01 00:30:30+08"]
(1 row)
```

- **tintervalend(tinterval)**
Description: Returns the end time of tinterval.
Return type: abstime
Example:

```
gaussdb=# SELECT tintervalend(['Sep 4, 1983 23:59:12' 'Oct4, 1983 23:59:12']);
tintervalend
-----
1983-10-04 23:59:12+08
(1 row)
```
- **tintervalrel(tinterval)**
Description: Calculates and returns the relative time of **tinterval**.
Return type: reltime
Example:

```
gaussdb=# SELECT tintervalrel(['Sep 4, 1983 23:59:12' 'Oct4, 1983 23:59:12']);
tintervalrel
-----
1 mon
(1 row)
```
- **smalldatetime_ge**
Description: Determines whether the first parameter is greater than the second.
Parameter: smalldatetime, smalldatetime
Return type: Boolean
- **smalldatetime_cmp**
Description: Compares two smalldatetime values to check whether they are the same.
Parameter: smalldatetime, smalldatetime
Return type: integer
- **smalldatetime_eq**
Description: Compares two smalldatetime values to check whether they are the same.
Parameter: smalldatetime, smalldatetime
Return type: Boolean
- **smalldatetime_gt**
Description: Determines whether the first parameter is less than the second parameter.
Parameter: smalldatetime, smalldatetime
Return type: Boolean
- **smalldatetime_hash**
Description: Calculates the hash value corresponding to a timestamp.
Parameter: smalldatetime
Return type: integer
- **smalldatetime_in**
Description: Inputs a timestamp.

Parameter: cstring, oid, integer

Return type: smalldatetime

- `smalldatetime_larger`
Description: Returns a larger timestamp.
Parameter: smalldatetime, smalldatetime
Return type: smalldatetime
- `smalldatetime_le`
Description: Determines whether the first parameter is less than the second parameter.
Parameter: smalldatetime, smalldatetime
Return type: Boolean
- `smalldatetime_lt`
Description: Determines whether the first parameter is greater than the second.
Parameter: smalldatetime, smalldatetime
Return type: Boolean
- `smalldatetime_ne`
Description: Compares two timestamps to check whether they are different.
Parameter: smalldatetime, smalldatetime
Return type: Boolean
- `smalldatetime_out`
Description: Converts a timestamp into the external form.
Parameter: smalldatetime
Return type: cstring
- `smalldatetime_send`
Description: Converts a timestamp to the binary format.
Parameter: smalldatetime
Return type: bytea
- `smalldatetime_smaller`
Description: Returns a smaller smalldatetime.
Parameter: smalldatetime, smalldatetime
Return type: smalldatetime
- `smalldatetime_to_abstime`
Description: Converts smalldatetime to abstime.
Parameter: smalldatetime
Return type: abstime
- `smalldatetime_to_time`
Description: Converts smalldatetime to time.
Parameter: smalldatetime

Return type: time without time zone

- `smalldatetime_to_timestamp`

Description: Converts `smalldatetime` to `timestamp`.

Parameter: `smalldatetime`

Return type: `timestamp` without time zone

- `smalldatetime_to_timestamptz`

Description: Converts `smalldatetime` to `timestamptz`.

Parameter: `smalldatetime`

Return type: `timestamp` with time zone

- `smalldatetime_to_varchar2`

Description: Converts `smalldatetime` to `varchar2`.

Parameter: `smalldatetime`

Return type: `character varying`

 **NOTE**

There are multiple methods for obtaining the current time. Select an appropriate API based on the actual service scenario.

1. The following APIs return values based on the start time of the current transaction:

```

CURRENT_DATE
CURRENT_TIME CURRENT_TIME(precision)
CURRENT_TIMESTAMP(precision)
LOCALTIME
LOCALTIMESTAMP
LOCALTIME(precision)
LOCALTIMESTAMP(precision)
transaction_timestamp()
now()
    
```

The values transferred by **CURRENT_TIME** and **CURRENT_TIMESTAMP(precision)** contain time zone information. The values transferred by **LOCALTIME** and **LOCALTIMESTAMP** do not contain time zone information. **CURRENT_TIME**, **LOCALTIME**, and **LOCALTIMESTAMP** can specify a precision parameter, which rounds the seconds field of the result to the decimal place. If there is no precision parameter, the result is given the full precision that can be obtained.

Because these functions all return results by the start time of the current transaction, their values do not change throughout the transaction. We think this is a feature with the purpose to allow a transaction to have a consistent concept at the "current" time, so that multiple modifications in the same transaction can maintain the same timestamp.

transaction_timestamp() is equivalent to **CURRENT_TIMESTAMP(precision)**, indicating the start time of the transaction where the current statement is located. **now()** is equivalent to **transaction_timestamp()**.

2. The following APIs return the start time of the current statement:

```
statement_timestamp()
```

statement_timestamp() returns the start time of the current statement (more accurately, the time when the last instruction is received from the client). The return values of **statement_timestamp()** and **transaction_timestamp()** are the same during the execution of the first instruction of a transaction, but may be different in subsequent instructions.

3. The following APIs return the actual current time when the function is called:

```
clock_timestamp()
timeofday()
```

clock_timestamp() returns the actual current time, and its value changes even in the same SQL command. Similar to **clock_timestamp()**, **timeofday()** also returns the actual current time. However, the result of **timeofday()** is a formatted text string instead of a timestamp with time zone information.

[Table 7-31](#) shows the templates for truncating date/time values.

Table 7-31 Truncating date/time values

Item	Format	Description
Microsecond	MICROSECON	Truncates date/time values, accurate to the microsecond (000000–999999).
	US	
	USEC	
	USECOND	

Item	Format	Description
Millisecond	MILLISECON	Truncates date/time values, accurate to the millisecond (000–999).
	MS	
	MSEC	
	MSECOND	
Second	S	Truncates date/time values, accurate to the second (00–59).
	SEC	
	SECOND	
Minute	M	Truncates date/time values, accurate to the minute (00–59).
	MI	
	MIN	
	MINUTE	
Hour	H	Truncates date/time values, accurate to the hour (00–23).
	HH	
	HOUR	
	HR	
Day	D	Truncates date/time values, accurate to the day (01-01 to 12-31)
	DAY	
	DD	
	DDD	
	J	
Week	W	Truncates date/time values, accurate to the week (the first day of the current week).
	WEEK	
Month	MM	Truncates date/time values, accurate to the month (the first day of the current month).
	MON	
	MONTH	
Quarter	Q	Truncates date/time values, accurate to the quarter (the first day of the current quarter).
	QTR	
	QUARTER	
Year	Y	Truncates date/time values, accurate to the year (the first day of the current year).
	YEAR	

Item	Format	Description
	YR	
	YYYY	
Decade	DEC	Truncates date/time values, accurate to the decade (the first day of the current decade).
	DECADE	
Century	C	Truncates date/time values, accurate to the century (the first day of the current century).
	CC	
	CENT	
	CENTURY	
Millennium	MIL	Truncates date/time values, accurate to the millennium (the first day of the current millennium).
	MILLENNIA	
	MILLENNIUM	

Table 7-32 Parameters for time truncation and rounding

Item	Format	Description
Minute	M	Truncated or rounded off, accurate to minute (00-59).
	MI	
	MIN	
	MINUTE	
Hour	H	Truncated or rounded off, accurate to hour (00-23).
	HH	
	HOUR	
	HR	
	HH12	
	HH24	
Day	DD	Truncated or rounded off, accurate to day (01-01 to 12-31).
	DDD	
	J	
ISO week	IW	Truncated or rounded off, accurate to week (the first day of the week is Monday).

Item	Format	Description
Week	DAY	Truncated or rounded off, accurate to week (the first day of the week is Sunday).
	DY	
	D	
Week of the month	W	Truncated or rounded off, accurate to week (the first day of the week is the first day of the month).
Week of the year	WW	Truncated or rounded off, accurate to week (the first day of the week is the first day of the year).
Month	MM	Truncated or rounded off, accurate to month (the first day of the month).
	MON	
	MONTH	
	RM	
Quarter	Q	Truncated or rounded off, accurate to quarter (the first day of the quarter).
	QTR	
	QUARTER	
Year	Y	Truncated or rounded off, accurate to year (the first day of the current year).
	YEAR	
	YR	
	YYYY	
	SYYYY	
	YYY	
	YY	
	SYEAR	
Decade	DEC	Truncated or rounded off, accurate to decade (the first day of the current decade).
	DECADE	
Century	C	Truncated or rounded off, accurate to the century (the first day of the century).
	CC	
	CENT	
	CENTURY	
	SCC	

Item	Format	Description
Millennium	MIL	Truncated or rounded off, accurate to millennium (the first day of the millennium).
	MILLENNIA	
	MILLENNIUM	

 NOTE

The behaviors of [Table 7-32](#) are valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s1** in an ORA-compatible database.

- `timestamp_diff(text, timestamp, timestamp)`
Description: Calculates the difference between two timestamps and truncates the difference to the precision specified by text.

Return type: int64

Example:

```
gaussdb=# SELECT timestamp_diff('year','2018-01-01','2020-04-01');
timestamp_diff
-----
          2
(1 row)
gaussdb=# SELECT timestamp_diff('month','2018-01-01','2020-04-01');
timestamp_diff
-----
         27
(1 row)
gaussdb=# SELECT timestamp_diff('quarter','2018-01-01','2020-04-01');
timestamp_diff
-----
          9
(1 row)
gaussdb=# SELECT timestamp_diff('week','2018-01-01','2020-04-01');
timestamp_diff
-----
        117
(1 row)
gaussdb=# SELECT timestamp_diff('day','2018-01-01','2020-04-01');
timestamp_diff
-----
        821
(1 row)
gaussdb=# SELECT timestamp_diff('hour','2018-01-01 10:10:10','2018-01-01 12:12:12');
timestamp_diff
-----
          2
(1 row)
gaussdb=# SELECT timestamp_diff('minute','2018-01-01 10:10:10','2018-01-01 12:12:12');
timestamp_diff
-----
        122
(1 row)
gaussdb=# SELECT timestamp_diff('second','2018-01-01 10:10:10','2018-01-01 10:12:12');
timestamp_diff
-----
        122
(1 row)
gaussdb=# SELECT timestamp_diff('microsecond','2018-01-01 10:10:10','2018-01-01 10:12:12');
timestamp_diff
-----
```

```
122000000
(1 row)
```

TIMESTAMPDIFF

- **TIMESTAMPDIFF**(*unit*, *timestamp_expr1*, *timestamp_expr2*)

The **timestampdiff** function returns the result of **timestamp_expr2** - **timestamp_expr1** in the specified unit. **timestamp_expr1** and **timestamp_expr2** must be value expressions of the **timestamp**, **timestamptz**, or **date** type. **unit** specifies the unit of the difference between two dates.

This function is equivalent to **timestamp_diff(text, timestamp, timestamp)**.

NOTE

This function is valid only when GaussDB is compatible with MySQL (that is, **dbcompatibility** is set to **'MYSQL'**).

- **year**

Year.

```
gaussdb=# SELECT TIMESTAMPDIFF(YEAR, '2018-01-01', '2020-01-01');
timestamp_diff
-----
                2
(1 row)
```

- **quarter**

Quarter.

```
gaussdb=# SELECT TIMESTAMPDIFF(QUARTER, '2018-01-01', '2020-01-01');
timestamp_diff
-----
                8
(1 row)
```

- **month**

Month.

```
gaussdb=# SELECT TIMESTAMPDIFF(MONTH, '2018-01-01', '2020-01-01');
timestamp_diff
-----
                24
(1 row)
```

- **week**

Week.

```
gaussdb=# SELECT TIMESTAMPDIFF(WEEK, '2018-01-01', '2020-01-01');
timestamp_diff
-----
               104
(1 row)
```

- **day**

Day.

```
gaussdb=# SELECT TIMESTAMPDIFF(DAY, '2018-01-01', '2020-01-01');
timestamp_diff
-----
               730
(1 row)
```

- **hour**

Hour.

```
gaussdb=# SELECT TIMESTAMPDIF(HOUR, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
timestamp_diff
-----
          1
(1 row)
```

- minute

Minute.

```
gaussdb=# SELECT TIMESTAMPDIF(MINUTE, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
timestamp_diff
-----
          61
(1 row)
```

- second

Second.

```
gaussdb=# SELECT TIMESTAMPDIF(SECOND, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
timestamp_diff
-----
         3661
(1 row)
```

- microseconds

The second field, including fractional parts, is multiplied by 1,000,000.

```
gaussdb=# SELECT TIMESTAMPDIF(MICROSECOND, '2020-01-01 10:10:10.000000', '2020-01-01
10:10:10.111111');
timestamp_diff
-----
        111111
(1 row)
```

- timestamp_expr with the time zone

```
gaussdb=# SELECT TIMESTAMPDIF(HOUR,'2020-05-01 10:10:10-01','2020-05-01 10:10:10-03');
timestamp_diff
-----
          2
(1 row)
```

EXTRACT

- **EXTRACT(*field* FROM *source*)**

The **extract** function retrieves fields such as year or hour from date/time values. **source** must be a value expression of type **timestamp**, **time**, or **interval**. (Expressions of type **date** are cast to **timestamp** and can therefore be used as well.) **field** is an identifier or string that selects what field to extract from the source value. The **extract** function returns values of type **double precision**. The following are valid **field** names:

- century

Century.

The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number 0. You go from **-1** century to **1** century.

Example:

```
gaussdb=# SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
date_part
-----
         20
(1 row)
```

- **day**
 - For **timestamp** values, the day (of the month) field (1–31)

```
gaussdb=# SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
      16
(1 row)
```
 - For **interval** values, the number of days

```
gaussdb=# SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
date_part
-----
      40
(1 row)
```
- **decade**
 Year divided by 10

```
gaussdb=# SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
     200
(1 row)
```
- **dow**
 Day of the week as Sunday (0) to Saturday (6)

```
gaussdb=# SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
       5
(1 row)
```
- **doy**
 Day of the year (1–365 or 366)

```
gaussdb=# SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
       47
(1 row)
```
- **epoch**
 - For **timestamp with time zone** values, the number of seconds since 1970-01-01 00:00:00-00 UTC (can be negative).
 For **date** and **timestamp** values, the number of seconds since 1970-01-01 00:00:00-00 local time.
 For **interval** values, the total number of seconds in the interval.

```
gaussdb=# SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16
20:38:40.12-08');
date_part
-----
982384720.12
(1 row)
gaussdb=# SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
date_part
-----
      442800
(1 row)
```
 - Way to convert an epoch value back to a timestamp

```
gaussdb=# SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 * INTERVAL '1
second' AS RESULT;
result
-----
2001-02-17 12:38:40.12+08
(1 row)
```

- hour

Hour (0–23)

```
gaussdb=# SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
      20
(1 row)
```

- isodow

Day of the week (1–7)

Monday is 1 and Sunday is 7.

 **NOTE**

This is identical to **dow** except for Sunday.

```
gaussdb=# SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
date_part
-----
       7
(1 row)
```

- isoyear

The ISO 8601 year that the date falls in (not applicable to intervals).

Each ISO year begins with the Monday of the week containing January 4, so in early January or late December the ISO year may be different from the Gregorian year. See [week](#) for more information.

```
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
date_part
-----
     2005
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-01 00:00:40');
date_part
-----
      52
(1 row)
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
date_part
-----
     2006
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-02 00:00:40');
date_part
-----
       1
(1 row)
```

- microseconds

The second field, including fractional parts, is multiplied by 1,000,000.

```
gaussdb=# SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
date_part
-----
28500000
(1 row)
```

- millennium

Millennium.

Years in the 1900s are in the second millennium. The third millennium started from January 1, 2001.

```
gaussdb=# SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
```

- ```

3
(1 row)

```
- **milliseconds**

Second field, including fractional parts, is multiplied by 1000. Note that this includes full seconds.

```

gaussdb=# SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
date_part

28500
(1 row)

```
  - **minute**

Minute (0–59).

```

gaussdb=# SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

38
(1 row)

```
  - **month**

For **timestamp** values, the specific month in the year (1–12).

```

gaussdb=# SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

2
(1 row)

```

For **interval** values, the number of months, modulo 12 (0–11).

```

gaussdb=# SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
date_part

1
(1 row)

```
  - **quarter**

Quarter of the year (1–4) that the date is in.

```

gaussdb=# SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

1
(1 row)

```
  - **second**

Second field, including fractional parts (0–59).

```

gaussdb=# SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');
date_part

28.5
(1 row)

```
  - **timezone**

Time zone offset from UTC, measured in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.
  - **timezone\_hour**

Hour component of the time zone offset.
  - **timezone\_minute**

Minute component of the time zone offset.
  - **week**

Number of the week of the year that the day is in. By definition (ISO 8601), the first week of a year contains January 4 of that year. (The ISO-8601 week

starts on Monday.) In other words, the first Thursday of a year is in week 1 of that year.

Because of this, it is possible for early January dates to be part of the 52nd or 53rd week of the previous year, and late December dates to be part of the 1st week of the next year. For example, 2006-01-01 is the 52nd week of 2005, and 2006-01-02 is the first week of 2006. You are advised to use the columns **isoyear** and **week** together to ensure consistency.

```
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
date_part

 2005
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-01 00:00:40');
date_part

 52
(1 row)
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
date_part

 2006
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-02 00:00:40');
date_part

 1
(1 row)
```

- year

Year field.

```
gaussdb=# SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

 2001
(1 row)
```

## date\_part

The **date\_part** function is modeled on the traditional Ingres equivalent to the SQL-standard function **extract**:

**date\_part('field', source)**

Note that here the **field** parameter needs to be a string value, not a name. The valid field names for **date\_part** are the same as for **extract**. For details, see [EXTRACT](#).

Example:

```
gaussdb=# SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
date_part

 16
(1 row)
gaussdb=# SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
date_part

 4
(1 row)
```

**Table 7-33** specifies the schema for formatting date and time values.

**Table 7-33** Schema for formatting date and time

| Category              | Format     | Description                                                                                                                                                                                 |
|-----------------------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Hour                  | HH         | Number of hours in one day (01-12)                                                                                                                                                          |
|                       | HH12       | Number of hours in one day (01-12)                                                                                                                                                          |
|                       | HH24       | Number of hours in one day (00-23)                                                                                                                                                          |
| Minute                | MI         | Minute (00-59)                                                                                                                                                                              |
| Second                | SS         | Second (00-59)                                                                                                                                                                              |
|                       | FF         | Microsecond (000000-999999)                                                                                                                                                                 |
|                       | FF1        | Microsecond (0-9)                                                                                                                                                                           |
|                       | FF2        | Microsecond (00-99)                                                                                                                                                                         |
|                       | FF3        | Microsecond (000-999)                                                                                                                                                                       |
|                       | FF4        | Microsecond (0000-9999)                                                                                                                                                                     |
|                       | FF5        | Microsecond (00000-99999)                                                                                                                                                                   |
|                       | FF6        | Microsecond (000000-999999)                                                                                                                                                                 |
|                       | SSSSS      | Second after midnight (0-86399)                                                                                                                                                             |
| Morning and afternoon | AM or A.M. | Morning identifier                                                                                                                                                                          |
|                       | PM or P.M. | Afternoon identifier                                                                                                                                                                        |
| Year                  | Y,YYY      | Year with comma (with four digits or more)                                                                                                                                                  |
|                       | SYYYY      | Year with four digits BC                                                                                                                                                                    |
|                       | YYYY       | Year (with four digits or more)                                                                                                                                                             |
|                       | YYY        | Last three digits of a year                                                                                                                                                                 |
|                       | YY         | Last two digits of a year                                                                                                                                                                   |
|                       | Y          | Last one digit of a year                                                                                                                                                                    |
|                       | IYYY       | ISO year (with four digits or more)                                                                                                                                                         |
|                       | IYY        | Last three digits of an ISO year                                                                                                                                                            |
|                       | IY         | Last two digits of an ISO year                                                                                                                                                              |
|                       | I          | Last one digit of an ISO year                                                                                                                                                               |
|                       | RR         | Last two digits of a year (A year of the 20th century can be stored in the 21st century.)                                                                                                   |
|                       | RRRR       | Capable of receiving a year with four digits or two digits. If there are 2 digits, the value is the same as the returned value of RR. If there are 4 digits, the value is the same as YYYY. |

| Category    | Format                                                                               | Description                                                                               |
|-------------|--------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
|             | <ul style="list-style-type: none"> <li>• BC or B.C.</li> <li>• AD or A.D.</li> </ul> | Era indicator Before Christ (BC) and After Christ (AD)                                    |
| Month       | MONTH                                                                                | Full spelling of a month in uppercase (9 characters are filled in if the value is empty.) |
|             | MON                                                                                  | Month in abbreviated format in uppercase (with three characters)                          |
|             | MM                                                                                   | Month (01-12)                                                                             |
|             | RM                                                                                   | Month in Roman numerals (I-XII; I=JAN) and uppercase                                      |
| Day         | DAY                                                                                  | Full spelling of a date in uppercase (9 characters are filled in if the value is empty.)  |
|             | DY                                                                                   | Day in abbreviated format in uppercase (with three characters)                            |
|             | DDD                                                                                  | Day in a year (001-366)                                                                   |
|             | DD                                                                                   | Day in a month (01-31)                                                                    |
|             | D                                                                                    | Day in a week (1-7).                                                                      |
| Week        | W                                                                                    | Week in a month (1-5) (The first week starts from the first day of the month.)            |
|             | WW                                                                                   | Week in a year (1-53) (The first week starts from the first day of the year.)             |
|             | IW                                                                                   | Week in an ISO year (The first Thursday is in the first week.)                            |
| Century     | CC                                                                                   | Century (with two digits) (The 21st century starts from 2001-01-01.)                      |
| Julian date | J                                                                                    | Julian date (starting from January 1 of 4712 BC)                                          |
| Quarter     | Q                                                                                    | Quarter                                                                                   |

When **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1** in an ORA-compatible database, the date and time will be formatted in the specified format.

**Table 7-34** Patterns for formatting date and time

| Item    | Format | Description                                              |
|---------|--------|----------------------------------------------------------|
| Century | SCC    | Century. A hyphen (-) will be displayed before BC years. |

| Item        | Format | Description                                                                             |
|-------------|--------|-----------------------------------------------------------------------------------------|
| Year        | SYYYY  | Returns a numeric year. A hyphen (-) will be displayed before BC years.                 |
|             | RR     | Returns the two-digit year of a date.                                                   |
|             | RRRR   | Returns the four-digit year of a date.                                                  |
|             | YEAR   | Returns the year of the character type.                                                 |
|             | SYEAR  | Returns the year of the character type. A hyphen (-) will be displayed before BC years. |
| Date Format | DL     | Returns the date in the specified long date format.                                     |
|             | DS     | Returns the date in the specified short date format.                                    |
|             | TS     | Returns the time in the specified time format.                                          |
| Second      | FF7    | Microsecond (0000000-9999990)                                                           |
|             | FF8    | Microsecond (00000000-99999900)                                                         |
|             | FF9    | Microsecond (000000000-999999000)                                                       |

 **NOTE**

In the table, the rules for RR to calculate years are as follows:

- If the range of the input two-digit year is between 00 and 49:
  - If the last two digits of the current year are between 00 and 49, the first two digits of the returned year are the same as the first two digits of the current year.
  - If the last two digits of the current year are between 50 and 99, the first two digits of the returned year equal to the first two digits of the current year plus 1.
- If the range of the input two-digit year is between 50 and 99:
  - If the last two digits of the current year are between 00 and 49, the first two digits of the returned year are equal to the first two digits of the current year minus 1.
  - If the last two digits of the current year are between 50 and 99, the first two digits of the returned year are the same as the first two digits of the current year.

 NOTE

In the scenario where this function is in an ORA-compatible database, the value of **a\_format\_version** is **10c**, and that of **a\_format\_dev\_version** is **s1**:

- The **to\_date** and **to\_timestamp** functions support the FX pattern (the input strictly corresponds to a pattern) and the X pattern (decimal point).
- The input pattern can appear only once, indicating that the patterns of the same information cannot appear at the same time. For example, SYYYY and BC cannot be used together.
- The pattern is case-insensitive.
- You are advised to use a separator between the input and the pattern. Otherwise, the behavior may not be compatible with database O.

## 7.5.9 Type Conversion Functions

### Type Conversion Functions

- **cash\_words(money)**

Description: Type conversion function, which converts money into text.

Example:

```
gaussdb=# SELECT cash_words('1.23');
 cash_words

One dollar and twenty three cents
(1 row)
```

- **cast(x as y [DEFAULT z ON CONVERSION ERROR][,fmt])**

Description: Converts *x* into the type specified by *y*.

- **DEFAULT z ON CONVERSION ERROR:** This parameter is optional. If the attempt to convert *x* to type *y* fails, *z* is converted to type *y* by default.
- *fmt*: This parameter is optional, which can be specified when *y* is one of the following data types:
  - int1/int2/int4/int8/int16/float4/float8/numeric: The function of the optional parameter *fmt* is the same as that in the **to\_number(expr [,fmt])** function.
  - date/timestamp/timestamp with time zone: The function of the optional parameter *fmt* is the same as that in the **to\_date(string [,fmt])** / **to\_timestamp(string [,fmt])** / **to\_timestamp\_tz(string [,fmt])** function.

Example:

```
gaussdb=# SELECT cast('22-ocX-1997' as timestamp DEFAULT '22-oct-1997' ON CONVERSION
ERROR, 'DD-Mon-YYYY');
 timestamp

1997-10-22 00:00:00
(1 row)
```

 NOTE

The **DEFAULT z ON CONVERSION ERROR** and *fmt* syntaxes are supported only when **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1** in an ORA-compatible database.

- **hextoraw(raw)**

Description: Converts a string in hexadecimal format into binary format.

Return type: raw

Example:

```
gaussdb=# SELECT hextoraw('7D');
 hextoraw

 7D
(1 row)
```

- numtoday(numeric)

Description: Converts values of the number type into the timestamp of the specified type.

Return type: timestamp

Example:

```
gaussdb=# SELECT numtoday(2);
 numtoday

 2 days
(1 row)
```

- rawtohex(string)

Description: Converts a string in binary format into hexadecimal format.

The result is the ACSII code of the input characters in hexadecimal format.

Return type: varchar

Example:

```
gaussdb=# SELECT rawtohex('1234567');
 rawtohex

 31323334353637
(1 row)
```

- to\_blob(raw)

Description: Converts the RAW type to the BLOB type.

Return type: BLOB

Example:

```
gaussdb=# SELECT to_blob('0AADD343CDBBD'::RAW(10));
 to_blob

 00AADD343CDBBD
(1 row)
```

#### NOTE

The **to\_blob** function is supported only when the value of **a\_format\_version** is **10c** and the value of **a\_format\_dev\_version** is **s2**.

- to\_bigint(varchar)

Description: Converts the character type to the bigint type.

Return type: bigint

Example:

```
gaussdb=# SELECT to_bigint('123364545554455');
 to_bigint

 123364545554455
(1 row)
```

- to\_binary\_double(expr)

Description: Converts an expression to a value of the float8 type.

**expr**: supports the number, float4, and float8 data types and character strings that can be implicitly converted to numeric types.

Return type: float8

Example:

```
gaussdb=# SELECT to_binary_double('12345678');
to_binary_double

12345678
(1 row)
```

- to\_binary\_double(expr, fmt)

Description: Converts an expression to a value of the float8 type after format matching.

**expr/fmt:** supports character strings of the char, nchar, varchar2, and nvarchar2 types. The **expr** also supports numeric types that can be implicitly converted to character types.

Return type: float8

Example:

```
gaussdb=# SELECT to_binary_double('1,2,3', '9,9,9');
to_binary_double

123
(1 row)
```

- to\_binary\_double(expr default return\_value on conversion error)

Description: Converts an expression to a value of the float8 type. If the conversion fails, the default value **return\_value** is returned.

**expr:** supports the number, float4, and float8 data types and numeric types that can be implicitly converted to character strings.

Note: If **expr** is not of the numeric or character type, an error is reported.

Return type: float8

Example:

```
gaussdb=# SELECT to_binary_double(1e2 default 12 on conversion error);
to_binary_double

100
(1 row)

gaussdb=# SELECT to_binary_double('aa' default 12 on conversion error);
to_binary_double

12
(1 row)
```

- to\_binary\_double(expr default return\_value on conversion error, fmt)

Description: Converts an expression to a value of the float8 type after format matching. If the expression fails to be converted, the default value **return\_value** is returned.

**expr/fmt:** supports the char, nchar, varchar2, and nvarchar2 types. **expr** also supports numeric types that can be implicitly converted to character types.

Return type: float8

Example:

```
gaussdb=# SELECT to_binary_double('12-' default 10 on conversion error, '99S');
to_binary_double

-12
(1 row)
```

```
gaussdb=# SELECT to_binary_double('aa-' default 12 on conversion error, '99S');
to_binary_double

 10
(1 row)
```

 **NOTE**

The to\_binary\_double function is supported only when the value of **a\_format\_version** is **10c** and the value of **a\_format\_dev\_version** is **s2** in an ORA-compatible database.

- to\_binary\_float(expr)

Description: Converts an expression to a value of the float4 type.

**expr**: supports the number, float4, and float8 data types and character strings that can be implicitly converted to numeric types.

Return type: float4

Example:

```
gaussdb=# SELECT to_binary_float('12345678');
to_binary_float

1.23457e+07
(1 row)
```

- to\_binary\_float(expr, fmt)

Description: Converts an expression to a value of the float4 type after format matching.

**expr/fmt**: supports character strings of the char, nchar, varchar2, and nvarchar2 types. The **expr** also supports numeric types that can be implicitly converted to character types.

Return type: float4

Example:

```
gaussdb=# SELECT to_binary_float('1,2,3', '9,9,9');
to_binary_float

 123
(1 row)
```

- to\_binary\_float(expr default return\_value on conversion error)

Description: Converts an expression to a value of the float4 type. If the conversion fails, the default value **return\_value** is returned.

**expr**: supports the number, float4, and float8 data types and numeric types that can be implicitly converted to character strings.

Note: If **expr** is not of the numeric or character type, an error is reported.

Return type: float4

Example:

```
gaussdb=# SELECT to_binary_float(1e2 default 12 on conversion error);
to_binary_float

 100
(1 row)

gaussdb=# SELECT to_binary_float('aa' default 12 on conversion error);
to_binary_float

 12
(1 row)
```

- to\_binary\_float(expr default return\_value on conversion error, fmt)

Description: Converts an expression to a value of the float4 type after format matching. If the expression fails to be converted, the default value **return\_value** is returned.

**expr/fmt**: supports character strings of the char, nchar, varchar2, and nvarchar2 types. **expr** also supports numeric types that can be implicitly converted to character types.

Return type: float4

Example:

```
gaussdb=# SELECT to_binary_float('12-' default 10 on conversion error, '99S');
to_binary_float
```

```

-12
(1 row)
```

```
gaussdb=# SELECT to_binary_float('aa-' default 12 on conversion error, '99S');
to_binary_float
```

```

10
(1 row)
```

 **NOTE**

The to\_binary\_float function is supported only when the value of **a\_format\_version** is **10c** and the value of **a\_format\_dev\_version** is **s2** in an ORA-compatible database.

- to\_char(datetime/interval [, fmt])

Description: Converts a DATETIME or INTERVAL value of the DATE/TIMESTAMP/TIMESTAMP WITH TIME ZONE/TIMESTAMP WITH LOCAL TIME ZONE type into the TEXT type according to the format specified by **fmt**.

- The optional parameter **fmt** allows for the following types: date, time, week, quarter, and century. Each type has a unique template. The templates can be combined together. Common templates include HH, MI, SS, YYYY, MM, and DD. For details, see [Table 7-33](#).
- A template may have a modification word. FM is a common modification word and is used to suppress the preceding zero or the following blank spaces.

Return type: text

Example:

```
gaussdb=# SELECT to_char(current_timestamp,'HH12:MI:SS');
to_char
```

```

10:19:26
(1 row)
```

```
gaussdb=# SELECT to_char(current_timestamp,'FMHH12:FMMI:FMSS');
to_char
```

```

10:19:46
(1 row)
```

- to\_char(double precision/real, text)

Description: Converts the values of the floating point type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_char(125.8::real, '999D99');
to_char
```

```

```

```
125.80
(1 row)
```

- `to_char` (numeric/smallint/integer/bigint/double precision/real[, fmt])

Descriptions: Converts an integer or a value in floating point format into a string in specified format.

- The optional parameter *fmt* allows for the following types: decimal point, group (thousand) separator, positive/negative sign and currency sign. Each type has a unique template. The templates can be combined together. Common templates include: 9, 0, millesimal sign (.), and decimal point (.). For details, see [Table 7-35](#).
- A template can have a modification word, similar to FM. However, FM does not suppress 0 which is output according to the template.
- Use the template X or x to convert an integer value into a string in hexadecimal format.

Return type: text

Example:

```
gaussdb=# SELECT to_char(1485,'9,999');
to_char

1,485
(1 row)
gaussdb=# SELECT to_char(1148.5,'9,999.999');
to_char

1,148.500
(1 row)
gaussdb=# SELECT to_char(148.5,'990999.909');
to_char

0148.500
(1 row)
gaussdb=# SELECT to_char(123,'XXX');
to_char

7B
(1 row)
```

**Table 7-35** fmt parameter of the number type

| Pattern    | Description                                       |
|------------|---------------------------------------------------|
| , (comma)  | Group (thousand) separator                        |
| . (period) | Decimal point                                     |
| \$         | \$ output to the specified position               |
| 0          | Value with leading zeros                          |
| 9          | Value with specified digits                       |
| B          | Space returned when the integer part is 0         |
| C          | Currency symbol (depending on the locale setting) |
| D          | Decimal point (depending on the locale setting)   |

| Pattern  | Description                                                           |
|----------|-----------------------------------------------------------------------|
| EEEE     | Scientific notation                                                   |
| G        | Group separator (depending on the locale setting)                     |
| L        | Currency symbol (depending on the locale setting)                     |
| MI       | Minus sign in the specified position (if the number is less than 0)   |
| PR       | Negative values in angle brackets                                     |
| RN       | Roman numeral (ranging from 1 to 3999)                                |
| S        | Signed number (depending on the locale setting)                       |
| TM       | Standard number in scientific notation                                |
| TM9      | Standard number in scientific notation                                |
| TME      | Standard number in scientific notation                                |
| U        | Currency symbol (depending on the locale setting)                     |
| V        | Decimal with specified number of digits shifted                       |
| PL       | Plus sign in the specified position (if the number is greater than 0) |
| SG       | Plus or minus sign in the specified position                          |
| TH or th | Ordinal number suffix                                                 |

 **NOTE**

This function supports the \$, C, TM, TM9, TME, and U patterns when **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1** in an ORA-compatible database. In addition, the **fmt** parameter cannot be set to **TH**, **PL**, or **SG** in this case.

- **to\_char(interval, text)**

Description: Converts the values of the time interval type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_char(interval '15h 2m 12s', 'HH24:MI:SS');
to_char

15:02:12
(1 row)
```

- **to\_char(integer, text)**

Description: Converts the values of the integer type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_char(125, '999');
to_char

125
(1 row)
```

- to\_char(set)

Description: Converts a value of the SET type to a string. The distributed system does not support the SET data type.

Return value: text

- to\_char(numeric, text)

Description: Converts the values of the numeric type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_char(-125.8, '999D99S');
to_char

125.80-
(1 row)
```

- to\_char (string)

Description: Converts the CHAR/VARCHAR/VARCHAR2/CLOB type into the TEXT type.

If this function is used to convert data of the CLOB type, and the value to be converted exceeds the value range of the target type, an error is returned.

Return type: text

Example:

```
gaussdb=# SELECT to_char('01110');
to_char

01110
(1 row)
```

- to\_nvarchar2(numeric)

Description: Converts to the nvarchar2 type.

Parameter: numeric

Return type: nvarchar2

- to\_char(timestamp, text)

Description: Converts the values of the timestamp type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_char(current_timestamp, 'HH12:MI:SS');
to_char

10:55:59
(1 row)
```

 NOTE

- When **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1**, the `to_char` function reports an error for the incorrect format (**fmt**).
- In non-compatible mode, the `to_char` function outputs the incorrect format (**fmt**) without change. For example, if the **fmt** is **FF10**, FF1 is matched for formatted output, and then **0** is output without change.

- `to_nchar (datetime/interval [, fmt])`

Description: Converts a DATETIME or INTERVAL value of the DATE/TIMESTAMP/TIMESTAMP WITH TIME ZONE/TIMESTAMP WITH LOCAL TIME ZONE type into the TEXT type according to the format specified by **fmt**.

- The optional parameter **fmt** allows for the following types: date, time, week, quarter, and century. Each type has a unique template. The templates can be combined together. Common templates include HH, MI, SS, YYYY, MM, and DD. For details, see [Table 7-33](#).
- A template may have a modification word. FM is a common modification word and is used to suppress the preceding zero or the following blank spaces.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar(current_timestamp,'HH12:MI:SS');
to_nchar

10:19:26
(1 row)
gaussdb=# SELECT to_nchar(current_timestamp,'FMHH12:FM MI:FMSS');
to_nchar

10:19:46
(1 row)
```

- `to_nchar(double precision/real, text)`

Description: Converts the values of the floating point type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar(125.8::real, '999D99');
to_nchar

125.80
(1 row)
```

- `to_nchar (numeric/smallint/integer/bigint/double precision/real[, fmt])`

Descriptions: Converts an integer or a value in floating point format into a string in specified format.

- The optional parameter *fmt* allows for the following types: decimal point, group (thousand) separator, positive/negative sign and currency sign. Each type has a unique template. The templates can be combined together. Common templates include: 9, 0, millesimal sign (,), and decimal point (.). For details, see [Table 7-35](#).
- A template can have a modification word, similar to FM. However, FM does not suppress 0 which is output according to the template.
- Use the template X or x to convert an integer value into a string in hexadecimal format.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar(1485,'9,999');
to_nchar

1,485
(1 row)
gaussdb=# SELECT to_nchar(1148.5,'9,999.999');
to_nchar

1,148.500
(1 row)
gaussdb=# SELECT to_nchar(148.5,'990999.909');
to_nchar

0148.500
(1 row)
gaussdb=# SELECT to_nchar(123,'XXX');
to_nchar

7B
(1 row)
```

 **NOTE**

This function is valid only when **sql\_compatibility** is set to 'ORA', **a\_format\_version** is set to **10c**, and **a\_format\_dev\_version** is set to **s2**. When the parameter is enabled, the \$, C, TM, TM9, TME, and U formats are supported. In addition, the **fmt** parameter cannot be set to **TH**, **PL**, or **SG** in this case.

- to\_nchar(interval, text)

Description: Converts the values of the time interval type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar(interval '15h 2m 12s', 'HH24:MI:SS');
to_nchar

15:02:12
(1 row)
```

- to\_nchar(integer, text)

Description: Converts the values of the integer type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar(125, '999');
to_nchar

125
(1 row)
```

- to\_nchar(set)

Description: Converts a value of the SET type to a string. The distributed system does not support the SET data type.

Return value: text

- to\_nchar(numeric, text)

Description: Converts the values of the numeric type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar(-125.8, '999D99S');
to_nchar

125.80-
(1 row)
```

- to\_nchar (string)

Description: Converts the CHAR/VARCHAR/VARCHAR2/CLOB type into the TEXT type.

If this function is used to convert data of the CLOB type, and the value to be converted exceeds the value range of the target type, an error is returned.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar('01110');
to_nchar

01110
(1 row)
```

- to\_nchar(timestamp, text)

Description: Converts the values of the timestamp type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar(current_timestamp, 'HH12:MI:SS');
to_nchar

10:55:59
(1 row)
```

- to\_clob(char/nchar/varchar/varchar2/nvarchar2/text/raw)

Description: Converts the RAW type or text character set type CHAR/NCHAR/VARCHAR/VARCHAR2/NVARCHAR2/TEXT into the CLOB type.

Return type: CLOB

Example:

```
gaussdb=# SELECT to_clob('ABCDEF'::RAW(10));
to_clob

ABCDEF
(1 row)
gaussdb=# SELECT to_clob('hello111'::CHAR(15));
to_clob

hello111
(1 row)
gaussdb=# SELECT to_clob('gauss123'::NCHAR(10));
to_clob

gauss123
(1 row)
gaussdb=# SELECT to_clob('gauss234'::VARCHAR(10));
to_clob

gauss234
(1 row)
gaussdb=# SELECT to_clob('gauss345'::VARCHAR2(10));
to_clob

```

```
gauss345
(1 row)
gaussdb=# SELECT to_clob('gauss456'::NVARCHAR2(10));
to_clob

gauss456
(1 row)
gaussdb=# SELECT to_clob('World222!'::TEXT);
to_clob

World222!
(1 row)
```

- `to_date(text)`

Description: Converts values of the text type into the timestamp in the specified format. Currently, only the following two formats are supported:

- Format 1: Date without separators, for example, 20150814. The value must contain the complete year, month, and day.
- Format 2: Date with separators, for example, 2014-08-14. The separator can be any non-digit character.

Return type: timestamp without time zone

Example:

```
gaussdb=# SELECT to_date('2015-08-14');
to_date

2015-08-14 00:00:00
(1 row)
```

- `to_date(text, text)`

Description: Converts the values of the string type into the dates in the specified format.

Return type: timestamp without time zone

Example:

```
gaussdb=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
to_date

2000-12-05 00:00:00
(1 row)
```

 **NOTE**

Case execution environment: The value of **a\_format\_version** is **10c**, the value of **a\_format\_dev\_version** is **s1**, and the value of **nls\_timestamp\_format** is **YYYY-MM-DD HH24:MI:SS**.

- `to_date(text [DEFAULT return_value ON CONVERSION ERROR [, fmt]])`

Description: Converts a string *text* into a value of the DATE type according to the format specified by *fmt*. If *fmt* is not specified, **a\_format\_version** is set to **10c**, and **a\_format\_dev\_version** is set to **s1**, the format specified by **nls\_timestamp\_format** is used for conversion. Otherwise, the fixed format (*fmt* = 'yyyy-mm-dd hh24-mi-ss') is used for conversion.

- **text**: any expression that can be calculated to CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT string. If null is entered, null is returned.
- **DEFAULT return\_value ON CONVERSION ERROR**: This parameter is optional, which can be used to specify the return value when *text* fails to be converted to the DATE type. The value of *return\_value* can be an expression or a bound variable that can be converted to the CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT type or null. The method of

converting *return\_value* to the DATE type is the same as that of converting *text* to the DATE type. If *return\_value* fails to be converted to the DATE type, an error is reported.

- *fmt*: This parameter is optional, which specifies the date and time model format of *text*. By default, *text* must comply with the default date format. If *fmt* is set to *J*, *text* must be an integer.

Return type: timestamp without time zone

Example:

```
gaussdb=# SELECT to_date('2015-08-14');
 to_date

2015-08-14 00:00:00
(1 row)
gaussdb=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
 to_date

2000-12-05 00:00:00
(1 row)
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s1';
SET
gaussdb=# SHOW nls_timestamp_format;
 nls_timestamp_format

DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)
gaussdb=# SELECT to_date('12-jan-2022' default '12-apr-2022' on conversion error);
 to_date

2022-01-12 00:00:00
(1 row)
gaussdb=# SELECT to_date('12-ja-2022' default '12-apr-2022' on conversion error);
 to_date

2022-04-12 00:00:00
(1 row)
gaussdb=# SELECT to_date('2022-12-12' default '2022-01-01' on conversion error, 'yyyy-mm-dd');
 to_date

2022-12-12 00:00:00
(1 row)
```

---

**CAUTION**

- The DEFAULT *return\_value* ON CONVERSION ERROR syntax is supported only when **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1**.
- When **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1**, the system may not report an error if the entered year exceeds 9999. For example, the result of **to\_date('99999-12-12', 'yyyy-mm-dd hh24:mi:ss')** is **9999-09-12 12:00:00**. If the value of year exceeds 9999, the number following 9999 will be parsed as the next **fmt**. This restriction also applies to **to\_timestamp**.

- 
- **to\_number** ( *expr* [, *fmt*])

Description: Converts **expr** into a value of the NUMBER type according to the specified format.

For details about the type conversion formats, see [Table 7-36](#).

If a hexadecimal string is converted into a decimal number, the hexadecimal string can include a maximum of 16 bytes if it is to be converted into an unsigned number.

During the conversion from a hexadecimal string to a decimal digit, the format string cannot have a character other than x or X. Otherwise, an error is reported.

Return type: number

Example:

```
gaussdb=# SELECT to_number('12,454.8-', '99G999D9S');
to_number

-12454.8
(1 row)
```

- `to_number(text, text)`

Description: Converts the values of the string type into the numbers in the specified format.

Return type: numeric

Example:

```
gaussdb=# SELECT to_number('12,454.8-', '99G999D9S');
to_number

-12454.8
(1 row)
```

- `to_number(expr [DEFAULT return_value ON CONVERSION ERROR [, fmt]])`

Description: Converts a string *expr* to a value of the numeric type based on the format specified by *fmt*. If *fmt* is not specified, *text* must be a character string that can be directly converted to numeric, for example, '123', '1e2'.

For details about the type conversion formats, see [Table 7-37](#).

- **expr**: an expression that can be converted into a CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT/INT/FLOAT string. If null is entered, null is returned.
- **DEFAULT return\_value ON CONVERSION ERROR**: This parameter is optional, which can be used to specify the return value when *expr* fails to be converted to the numeric type. Similar to *expr*, *return\_value* can be any type that can be converted to a character string. Similar to *expr*, *return\_value* is converted based on the format specified by *fmt*. The system checks whether *return\_value* fails to be converted. If *return\_value* fails to be converted, the function reports an error.
- **fmt**: This parameter is optional, which specifies the conversion format of **expr**.

If any input parameter is **NULL**, **NULL** is returned.

Return type: numeric

Example:

```
gaussdb=# SET a_format_version='10c';
gaussdb=# SET a_format_dev_version='s1';

gaussdb=# SELECT to_number('1e2');
to_number

100
```

```
(1 row)

gaussdb=# SELECT to_number('123.456');
to_number

123.456
(1 row)

gaussdb=# SELECT to_number('123', '999');
to_number

123
(1 row)

gaussdb=# SELECT to_number('123-', '999MI');
to_number

-123
(1 row)

gaussdb=# SELECT to_number('123' default '456-' on conversion error, '999MI');
to_number

-456
(1 row)
```

 NOTE

The DEFAULT *return\_value* ON CONVERSION ERROR syntax is supported only when **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1**.

- `to_timestamp(double precision)`

Description: Converts a Unix century into a timestamp.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT to_timestamp(1284352323);
to_timestamp

2010-09-13 12:32:03+08
(1 row)
```

- `to_timestamp(string [,fmt])`

Description: Converts a string into a value of the timestamp type according to the format specified by **fmt**. When **fmt** is not specified, perform the conversion according to the format specified by **nls\_timestamp\_format**.

In **to\_timestamp** in GaussDB:

- If the input year *YYYY* is 0, an error will be reported.
- If the input year *YYYY* is less than 0, specify *SYYYY* in **fmt**. The year with the value of *n* (an absolute value) BC will be output correctly.

Characters in the **fmt** must match the schema for formatting the data and time. Otherwise, an error is reported.

Return type: timestamp without time zone

Example:

```
gaussdb=# SHOW nls_timestamp_format;
nls_timestamp_format

DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

gaussdb=# SELECT to_timestamp('12-sep-2014');
to_timestamp
```

```

2014-09-12 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SS.FF');
to_timestamp

2010-09-12 14:10:10.123
(1 row)
gaussdb=# SELECT to_timestamp('-1','YYYYY');
to_timestamp

0001-01-01 00:00:00 BC
(1 row)
gaussdb=# SELECT to_timestamp('98','RR');
to_timestamp

1998-01-01 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('01','RR');
to_timestamp

2001-01-01 00:00:00
(1 row)
```

**NOTE**

1. When **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1**, **fmt** supports FF[7-9]. When FF[7-9] is used, and the length of the corresponding position in the string is less than or equal to the number following FF, then, the number can be converted. However, the maximum length of the final conversion result is six digits.
  2. The result returned by the `current_timestamp` function cannot be used as a string parameter.
- `to_timestamp(text [DEFAULT return_value ON CONVERSION ERROR [, fmt]])`  
Description: Converts a string *text* into a value of the DATE type according to the format specified by *fmt*. If *fmt* is not specified, **a\_format\_version** is set to **10c**, and **a\_format\_dev\_version** is set to **s1**, the format specified by **nls\_timestamp\_format** is used for conversion. Otherwise, the fixed format (*fmt* = 'yyyy-mm-dd hh24-mi-ss') is used for conversion.
    - *text*: any expression that can be calculated to CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT string. If null is entered, null is returned.
    - **DEFAULT return\_value ON CONVERSION ERROR**: This parameter is optional, which can be used to specify the return value when *text* fails to be converted to the DATE type. The value of *return\_value* can be an expression or a bound variable that can be converted to the CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT type or null. The method of converting *return\_value* to the timestamp type is the same as that of converting *text* to the timestamp type. If *return\_value* fails to be converted to the timestamp type, an error is reported.
    - *fmt*: This parameter is optional, which specifies the date and time model format of *text*. By default, *text* must comply with the default date format. If *fmt* is set to *J*, *text* must be an integer.

Return type: timestamp without time zone

Example:

```
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s1';
SET
gaussdb=# SELECT to_timestamp('11-Sep-11' DEFAULT '12-Sep-10 14:10:10.123000' ON
```

```
CONVERSION ERROR,'DD-Mon-YY HH24:MI:SS.FF');
to_timestamp

2011-09-11 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SSXFF');
to_timestamp

2010-09-12 14:10:10.123
(1 row)
```

#### NOTE

The DEFAULT *return\_value* ON CONVERSION ERROR syntax is supported only when **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1**.

- `to_timestamp(text, text)`

Description: Converts values of the string type into the timestamp of the specified type.

Return type: timestamp

Example:

```
gaussdb=# SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
to_timestamp

2000-12-05 00:00:00
(1 row)
```

- `to_timestamp_tz(text [,text])`

Description: Converts values of the string type into the timestamp of the specified type with the time zone.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT to_timestamp_tz('05 Dec 2001', 'DD Mon YYYY');
to_timestamp_tz

2001-12-05 00:00:00+08:00
(1 row)
```

#### NOTE

This function is valid only when the value of **a\_format\_version** is **10c** and that of **a\_format\_dev\_version** is **s1** in an ORA-compatible database.

- `to_timestamp_tz(string [DEFAULT return_value ON CONVERSION ERROR] [,fmt])`

Description: Converts a string into a value of the timestamp type with the time zone according to the format specified by **fmt**. When **fmt** is not specified, perform the conversion according to the format specified by **nls\_timestamp\_tz\_format**.

**DEFAULT return\_value ON CONVERSION ERROR:** This parameter is optional. If *string* fails to be converted to the timestamp type with time zone, *return\_value* is converted to the timestamp type with time zone.

*fmt*: This parameter is optional, which specifies the date and time model format of *string*. The setting of this parameter is the same as that in the `to_timestamp` function.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT to_timestamp_tz('05 DeX 2000' DEFAULT '05 Dec 2001' ON CONVERSION ERROR,
to_timestamp_tz

2001-12-05 00:00:00+08:00
(1 row)
```

 **NOTE**

This function is valid only when the value of **a\_format\_version** is **10c** and that of **a\_format\_dev\_version** is **s1** in an ORA-compatible database.

- to\_dsinterval(text)

Description: Converts characters to the interval type. SQL-compatible and ISO formats are supported.

Return type: interval

Example:

```
gaussdb=# SELECT to_dsinterval('12 1:2:3.456');
to_dsinterval

12 days 01:02:03.456
(1 row)
```

```
gaussdb=# SELECT to_dsinterval('P3DT4H5M6S');
to_dsinterval

3 days 04:05:06
(1 row)
```

 **NOTE**

This function is valid only when the value of **a\_format\_version** is **10c** and that of **a\_format\_dev\_version** is **s2** in an ORA-compatible database.

- to\_ymininterval(text)

Description: Converts characters to the interval type. SQL-compatible and ISO formats are supported.

Return type: interval

Example:

```
gaussdb=# SELECT to_ymininterval('1-1');
to_ymininterval

1 year 1 mon
(1 row)
```

```
gaussdb=# SELECT to_ymininterval('P13Y3M4DT4H2M5S');
to_ymininterval

13 years 3 mons
(1 row)
```

 **NOTE**

This function is valid only when the value of **a\_format\_version** is **10c** and that of **a\_format\_dev\_version** is **s2** in an ORA-compatible database.

**Table 7-36** Template patterns for numeric formatting

| Pattern | Description                 |
|---------|-----------------------------|
| 9       | Value with specified digits |

| Pattern    | Description                                                           |
|------------|-----------------------------------------------------------------------|
| 0          | Value with leading zeros                                              |
| Period (.) | Decimal point                                                         |
| Comma (,)  | Group (thousand) separator                                            |
| PR         | Negative values in angle brackets                                     |
| S          | Signed number (depending on the locale setting)                       |
| L          | Currency symbol (depending on the locale setting)                     |
| D          | Decimal point (depending on the locale setting)                       |
| G          | Group separator (depending on the locale setting)                     |
| MI         | Minus sign in the specified position (if the number is less than 0)   |
| PL         | Plus sign in the specified position (if the number is greater than 0) |
| SG         | Plus or minus sign in the specified position                          |
| RN         | Roman numeral (ranging from 1 to 3999)                                |
| TH or th   | Ordinal number suffix                                                 |
| V          | Decimal with specified number of digits shifted                       |
| x or X     | Hexadecimal-to-decimal conversion identifier                          |

**Table 7-37** Template patterns for to\_number formatting

| Pattern    | Description                                                                                                                   |
|------------|-------------------------------------------------------------------------------------------------------------------------------|
| 9          | Matches a digit. The number of digits "9" can be greater than or equal to that in the corresponding position in <i>expr</i> . |
| 0          | Strictly matches a digit. The number of digits "0" must be equal to that in <i>expr</i> .                                     |
| 5          | Matches digit 0 or 5.                                                                                                         |
| Period (.) | Decimal point in the specified position                                                                                       |
| Comma (,)  | Group (thousand) separator in the specified position. Multiple commas can be specified in <i>fmt</i> .                        |
| B          | No leading sign                                                                                                               |
| PR         | A negative value in angle brackets, or a positive value with no leading sign.                                                 |

| Pattern | Description                                                                                         |
|---------|-----------------------------------------------------------------------------------------------------|
| S       | A negative value with the leading minus sign (-) or a positive value with the leading plus sign (+) |
| MI      | A negative value with the leading minus sign (-) or a positive value with no leading sign           |
| \$      | Leading dollar sign                                                                                 |
| L       | Local currency symbol                                                                               |
| C       | Symbol of currency in the specified position (complying with the ISO standard)                      |
| U       | Dual-currency symbol                                                                                |
| D       | Decimal point (depending on the locale setting)                                                     |
| G       | Group separator (complying with the ISO standard). Multiple commas can be specified in <i>fmt</i> . |
| RN / rn | Roman numeral (ranging from 1 to 3999). <i>to_number</i> does not support this format.              |
| V       | <i>to_number</i> does not support this format.                                                      |
| X / x   | Conversion between hexadecimal and decimal                                                          |
| TM      | <i>to_number</i> does not support this format.                                                      |
| FM      | This format can be used only at the beginning of <i>fmt</i> .                                       |
| EEEE    | Conversion based on the scientific notation model                                                   |

 NOTE

When **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s1**, refer to this table for the setting of *fmt*. Otherwise, refer to the previous table. The *fmt* setting complying with the ISO standard is affected by the values of **LC\_MONETARY** and **LC\_NUMERIC** parameters.

- `cast_varchar2_to_raw_for_histogram(varchar2)`  
Description: Converts from the varchar2 type to the raw type.  
Return type: raw
- `abstime_text`  
Description: Converts abstime to text.  
Parameter: abstime  
Return type: text
- `abstime_to_smalldatetime`  
Description: Converts abstime to smalldatetime.  
Parameter: abstime  
Return type: smalldatetime

- `bigint_tid`  
Description: Converts bigint to tid.  
Parameter: bigint  
Return type: tid
- `bool_int1`  
Description: Converts Boolean to int1.  
Parameter: Boolean  
Return type: tinyint
- `bool_int2`  
Description: Converts Boolean to int2.  
Parameter: Boolean  
Return type: smallint
- `bool_int8`  
Description: Converts Boolean to int8.  
Parameter: Boolean  
Return type: bigint
- `bpchar_date`  
Description: Converts a string to a date.  
Parameter: character  
Return type: date
- `bpchar_float4`  
Description: Converts a string to float4.  
Parameter: character  
Return type: real
- `bpchar_float8`  
Description: Converts a string to float8.  
Parameter: character  
Return type: double precision
- `bpchar_int4`  
Description: Converts a string to int4.  
Parameter: character  
Return type: integer
- `bpchar_int8`  
Description: Converts a string to int8.  
Parameter: character  
Return type: bigint
- `bpchar_numeric`  
Description: Converts a string to numeric.  
Parameter: character  
Return type: numeric

- `bpchar_timestamp`  
Description: Converts a string to a timestamp.  
Parameter: character  
Return type: timestamp without time zone
- `bpchar_to_smalldatetime`  
Description: Converts a string to smalldatetime.  
Parameter: character  
Return type: smalldatetime
- `complex_array_in`  
Description: Converts the external `complex_array` type to the internal `anyarray` array type.  
Parameter: `cstring`, `oid`, `int2vector`  
Return type: `anyarray`
- `date_bpchar`  
Description: Converts the date type to `bpchar`.  
Parameter: date  
Return type: character
- `date_text`  
Description: Converts date to text.  
Parameter: date  
Return type: text
- `date_varchar`  
Description: Converts date to `varchar`.  
Parameter: date  
Return type: character varying
- `f4toi1`  
Description: Forcibly converts `float4` to `uint8`.  
Parameter: real  
Return type: `tinyint`
- `f8toi1`  
Description: Forcibly converts `float8` to `uint8`.  
Parameter: double precision  
Return type: `tinyint`
- `float4_bpchar`  
Description: Converts `float4` to `bpchar`.  
Parameter: real  
Return type: character
- `float4_text`  
Description: Converts `float4` to text.  
Parameter: real  
Return type: text

- float4\_varchar  
Description: Converts float4 to varchar.  
Parameter: real  
Return type: character varying
- float8\_bpchar  
Description: Converts float8 to bpchar.  
Parameter: double precision  
Return type: character
- float8\_interval  
Description: Converts float8 to interval.  
Parameter: double precision  
Return type: interval
- float8\_text  
Description: Converts float8 to text.  
Parameter: double precision  
Return type: text
- float8\_varchar  
Description: Converts float8 to varchar.  
Parameter: double precision  
Return type: character varying
- i1tof4  
Description: Converts uint8 to float4.  
Parameter: tinyint  
Return type: real
- i1tof8  
Description: Converts uint8 to float8.  
Parameter: tinyint  
Return type: double precision
- i1toi2  
Description: Converts uint8 to int16.  
Parameter: tinyint  
Return type: smallint
- i1toi4  
Description: Converts uint8 to int32.  
Parameter: tinyint  
Return type: integer
- i1toi8  
Description: Converts uint8 to int64.  
Parameter: tinyint  
Return type: bigint

- `i2toi1`  
Description: Converts int16 to uint8.  
Parameter: `smallint`  
Return type: `tinyint`
- `i4toi1`  
Description: Converts int32 to uint8.  
Parameter: `integer`  
Return type: `tinyint`
- `i8toi1`  
Description: Converts int64 to uint8.  
Parameter: `bigint`  
Return type: `tinyint`
- `int1_avg_accum`  
Description: Adds the second parameter of the uint8 type to the first parameter. The first parameter is an array of the bigint type.  
Parameter: `bigint[], tinyint`  
Return type: `bigint[]`
- `int1_bool`  
Description: Converts uint8 to Boolean.  
Parameter: `tinyint`  
Return type: `Boolean`
- `int1_bpchar`  
Description: Converts uint8 to bpchar.  
Parameter: `tinyint`  
Return type: `character`
- `int1_mul_cash`  
Description: Returns the product of a parameter of the int8 type and a parameter of the cash type. The return type is cash.  
Parameter: `tinyint, money`  
Return type: `money`
- `int1_numeric`  
Description: Converts uint8 to numeric.  
Parameter: `tinyint`  
Return type: `numeric`
- `int1_nvarchar2`  
Description: Converts uint8 to nvarchar2.  
Parameter: `tinyint`  
Return type: `nvarchar2`
- `int1_text`  
Description: Converts uint8 to text.  
Parameter: `tinyint`

- Return type: text
- int1\_varchar  
Description: Converts uint8 to varchar.  
Parameter: tinyint  
Return type: character varying
  - int1in  
Description: Converts a string into an unsigned 1-byte integer.  
Parameter:cstring  
Return type: tinyint
  - int1out  
Description: Converts an unsigned 1-byte integer into a string.  
Return type:cstring
  - int1up  
Description: Converts an input integer to an unsigned 1-byte integer.  
Parameter: tinyint  
Return type: tinyint
  - int2\_bool  
Description: Converts a two-byte signed integer to the Boolean type.  
Parameter: smallint  
Return type: Boolean
  - int2\_bpchar  
Description: Converts a signed two-byte integer to the bpchar type.  
Parameter: smallint  
Return type: character
  - int2\_text  
Description: Converts a signed two-byte integer to the text type.  
Parameter: smallint  
Return type: text
  - int2\_varchar  
Description: Converts a signed two-byte integer to the varchar type.  
Parameter: smallint  
Return type: character varying
  - int4\_bpchar  
Description: Converts a signed four-byte integer to bpchar.  
Parameter: integer  
Return type: character
  - int4\_text  
Description: Converts a signed four-byte integer to the text type.  
Parameter: integer  
Return type: text

- `int4_varchar`  
Description: Converts a signed four-byte integer into varchar.  
Parameter: integer  
Return type: character varying
- `int8_bool`  
Description: Converts a signed eight-byte integer to the Boolean value.  
Parameter: bigint  
Return type: Boolean
- `int8_bpchar`  
Description: Converts a signed eight-byte integer to the bpchar type.  
Parameter: bigint  
Return type: character
- `int8_text`  
Description: Converts a signed eight-byte integer to the text type.  
Parameter: bigint  
Return type: text
- `int8_varchar`  
Description: Converts a signed eight-byte integer to the varchar type.  
Parameter: bigint  
Return type: character varying
- `intervaltonum`  
Description: Converts the internal dats type date to numeric.  
Parameter: interval  
Return type: numeric
- `numeric_bpchar`  
Description: Converts numeric to bpchar.  
Parameter: numeric  
Return type: character
- `numeric_int1`  
Description: Converts numeric to a signed one-byte integer.  
Parameter: numeric  
Return type: tinyint
- `numeric_text`  
Description: Converts numeric to text.  
Parameter: numeric  
Return type: text
- `numeric_varchar`  
Description: Converts numeric to varchar.  
Parameter: numeric  
Return type: character varying

- `nvarchar2in`  
Description: Converts c string to varchar.  
Parameter: cstring, oid, integer  
Return type: nvarchar2
- `nvarchar2out`  
Description: Converts text into a c string.  
Parameter: nvarchar2  
Return type: cstring
- `nvarchar2send`  
Description: Converts varchar to binary.  
Parameter: nvarchar2  
Return type: bytea
- `oidvectorin_extend`  
Description: Converts a string to oidvector.  
Parameter: cstring  
Return type: oidvector\_extend
- `oidvectorout_extend`  
Description: Converts oidvector to a string.  
Parameter: oidvector\_extend  
Return type: cstring
- `oidvectorsend_extend`  
Description: Converts oidvector to a string.  
Parameter: oidvector\_extend  
Return type: bytea
- `reltime_text`  
Description: Converts reltime to text.  
Parameter: reltime  
Return type: text
- `text_date`  
Description: Converts the text type to the date type.  
Parameter: text  
Return type: date
- `text_float4`  
Description: Converts text to float4.  
Parameter: text  
Return type: real
- `text_float8`  
Description: Converts the text type to float8.  
Parameter: text  
Return type: double precision

- `text_int1`  
Description: Converts the text type to int1.  
Parameter: text  
Return type: tinyint
- `text_int2`  
Description: Converts the text type to the int2 type.  
Parameter: text  
Return type: smallint
- `text_int4`  
Description: Converts the text type to int4.  
Parameter: text  
Return type: integer
- `text_int8`  
Description: Converts the text type to the int8 type.  
Parameter: text  
Return type: bigint
- `text_numeric`  
Description: Converts the text type to the numeric type.  
Parameter: text  
Return type: numeric
- `text_timestamp`  
Description: Converts the text type to the timestamp type.  
Parameter: text  
Return type: timestamp without time zone
- `time_text`  
Description: Converts the time type to the text type.  
Parameter: time without time zone  
Return type: text
- `timestamp_text`  
Description: Converts the timestamp type to the text type.  
Parameter: timestamp without time zone  
Return type: text
- `timestamp_to_smalldatetime`  
Description: Converts the timestamp type to the smalldatetime type.  
Parameter: timestamp without time zone  
Return type: smalldatetime
- `timestamp_varchar`  
Description: Converts the timestamp type to varchar.  
Parameter: timestamp without time zone  
Return type: character varying

- `timestampz_to_smalldatetime`  
Description: Converts timestampz to smalldatetime.  
Parameter: timestamp with time zone  
Return type: smalldatetime
- `timestampzone_text`  
Description: Converts the timestampzone type to the text type.  
Parameter: timestamp with time zone  
Return type: text
- `timetz_text`  
Description: Converts the timetz type to the text type.  
Parameter: time with time zone  
Return type: text
- `to_integer`  
Description: Converts data to the integer type.  
Parameter: character varying  
Return type: integer
- `to_interval`  
Description: Converts to the interval type.  
Parameter: character varying  
Return type: interval
- `to_numeric`  
Description: Converts to the numeric type.  
Parameter: character varying  
Return type: numeric
- `to_text`  
Description: Converts to the text type.  
Parameter: smallint  
Return type: text
- `to_ts`  
Description: Converts to the ts type.  
Parameter: character varying  
Return type: timestamp without time zone
- `to_varchar2`  
Description: Converts to the varchar2 type.  
Parameter: timestamp without time zone  
Return type: character varying
- `varchar_date`  
Description: Converts varchar to date.  
Parameter: character varying  
Return type: date

- `varchar_float4`  
Description: Converts varchar to float4.  
Parameter: character varying  
Return type: real
- `varchar_float8`  
Description: Converts the varchar type to the float8 type.  
Parameter: character varying  
Return type: double precision
- `varchar_int4`  
Description: Converts the type from varchar to int4.  
Parameter: character varying  
Return type: integer
- `varchar_int8`  
Description: Converts the varchar type to the int8 type.  
Parameter: character varying  
Return type: bigint
- `varchar_numeric`  
Description: Converts varchar to numeric.  
Parameter: character varying  
Return type: numeric
- `varchar_timestamp`  
Description: Converts varchar to timestamp.  
Parameter: character varying  
Return type: timestamp without time zone
- `varchar2_to_smlldatetime`  
Description: Converts varchar2 to smlldatetime.  
Parameter: character varying  
Return type: smalldatetime
- `xidout4`  
Description: The xid output is a four-byte number.  
Parameter: xid32  
Return type:cstring
- `xidsend4`  
Description: Converts xid to the binary format.  
Parameter: xid32  
Return type:bytea
- `treat(expr AS [JSON | REF] schema.type)`  
Description: Converts expr to the type (JSON) specified by the keyword after AS.  
Return value type: JSON.  
Example:

```
gaussdb=# CREATE TABLE json_doc(data CLOB);
gaussdb=# INSERT INTO json_doc VALUES({'name':'a'});
gaussdb=# SELECT treat(data as json) FROM json_doc;
 json

{"name":"a"}
(1 row)
gaussdb=# DROP TABLE json_doc;
DROP TABLE
```

## Encoding Type Conversion

- `convert_to_nocase(text, text)`

Description: Converts a string into a specified encoding type.

Return type: bytea

Example:

```
gaussdb=# SELECT convert_to_nocase('12345', 'GBK');
convert_to_nocase

\x3132333435
(1 row)
```

## 7.5.10 Geometric Functions and Operators

### Geometric Operators

- `+`

Description: Translation

Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' + point '(2.0,0)' AS RESULT;
 result

(3,1),(2,0)
(1 row)
```

- `-e`

Description: Translation

Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' - point '(2.0,0)' AS RESULT;
 result

(-1,1),(-2,0)
(1 row)
```

- `*`

Description: Scaling out/Rotation

Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
 result

(2,2),(0,0)
(1 row)
```

- `/`

Description: Scaling in/Rotation

Example:

```
gaussdb=# SELECT box '((0,0),(2,2))' / point '(2.0,0)' AS RESULT;
 result
```

- ```
-----
(1,1),(0,0)
(1 row)
```

 - #

Description: Intersection of two figures

Example:

```
gaussdb=# SELECT box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))' AS RESULT;
result
-----
(1,1),(-1,-1)
(1 row)
```
 - #

Description: Number of paths or polygon vertexes

Example:

```
gaussdb=# SELECT # path '((1,0),(0,1),(-1,0))' AS RESULT;
result
-----
3
(1 row)
```
 - @-@

Description: Length or circumference

Example:

```
gaussdb=# SELECT @-@ path '((0,0),(1,0))' AS RESULT;
result
-----
2
(1 row)
```
 - @@

Description: Center of box

Example:

```
gaussdb=# SELECT @@ circle '((0,0),10)' AS RESULT;
result
-----
(0,0)
(1 row)
```
 - <->

Description: Distance between the two figures

Example:

```
gaussdb=# SELECT circle '((0,0),1)' <-> circle '((5,0),1)' AS RESULT;
result
-----
3
(1 row)
```
 - &&

Description: Overlaps? (One point in common makes this true.)

Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' && box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```
 - <<

Description: Is strictly left of (no common horizontal coordinate)?

Example:

```
gaussdb=# SELECT circle '((0,0),1)' << circle '((5,0),1)' AS RESULT;
result
-----
t
(1 row)
```

- >>

Description: Is strictly right of (no common horizontal coordinate)?

Example:

```
gaussdb=# SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
result
-----
t
(1 row)
```

- &<

Description: Does not extend to the right of?

Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' &< box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

- &>

Description: Does not extend to the left of?

Example:

```
gaussdb=# SELECT box '((0,0),(3,3))' &> box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

- <<|

Description: Is strictly below (no common horizontal coordinate)?

Example:

```
gaussdb=# SELECT box '((0,0),(3,3))' <<| box '((3,4),(5,5))' AS RESULT;
result
-----
t
(1 row)
```

- |>>

Description: Is strictly above (no common horizontal coordinate)?

Example:

```
gaussdb=# SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
result
-----
t
(1 row)
```

- &<|

Description: Does not extend above?

Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

- |>
Description: Does not extend below?
Example:

```
gaussdb=# SELECT box '((0,0),(3,3))' |> box '((0,0),(2,2))' AS RESULT;  
result  
-----  
t  
(1 row)
```
- <^
Description: Is below (allows touching)?
Example:

```
gaussdb=# SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;  
result  
-----  
t  
(1 row)
```
- >^
Description: Is above (allows touching)?
Example:

```
gaussdb=# SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))' AS RESULT;  
result  
-----  
t  
(1 row)
```
- ?#
Description: Intersect?
Example:

```
gaussdb=# SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;  
result  
-----  
t  
(1 row)
```
- ?-
Description: Is horizontal?
Example:

```
gaussdb=# SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;  
result  
-----  
t  
(1 row)
```
- ?-
Description: Are horizontally aligned?
Example:

```
gaussdb=# SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;  
result  
-----  
t  
(1 row)
```
- ?|
Description: Is vertical?
Example:

```
gaussdb=# SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;  
result
```

- ```

f
(1 row)
```
- ?|

Description: Are vertically aligned?

Example:

```
gaussdb=# SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;
result

t
(1 row)
```
  - ?-|

Description: Are perpendicular?

Example:

```
gaussdb=# SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;
result

t
(1 row)
```
  - ?||

Description: Are parallel?

Example:

```
gaussdb=# SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;
result

t
(1 row)
```
  - @>

Description: Contains?

Example:

```
gaussdb=# SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;
result

t
(1 row)
```
  - <@

Description: Contained in or on?

Example:

```
gaussdb=# SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;
result

t
(1 row)
```
  - ~=

Description: Specifies whether two figures are the same.

Example:

```
gaussdb=# SELECT polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' AS RESULT;
result

t
(1 row)
```

## Geometric Functions

- **area(object)**  
Description: Area calculation  
Return type: double precision  
Example:

```
gaussdb=# SELECT area(box '((0,0),(1,1)')) AS RESULT;
result

 1
(1 row)
```
- **center(object)**  
Description: Figure center calculation  
Return type: point  
Example:

```
gaussdb=# SELECT center(box '((0,0),(1,2)')) AS RESULT;
result

(0.5,1)
(1 row)
```
- **diameter(circle)**  
Description: Circle diameter calculation  
Return type: double precision  
Example:

```
gaussdb=# SELECT diameter(circle '((0,0),2.0)') AS RESULT;
result

 4
(1 row)
```
- **height(box)**  
Description: Vertical size of box  
Return type: double precision  
Example:

```
gaussdb=# SELECT height(box '((0,0),(1,1)')) AS RESULT;
result

 1
(1 row)
```
- **isclosed(path)**  
Description: A closed path?  
Return type: Boolean  
Example:

```
gaussdb=# SELECT isclosed(path '((0,0),(1,1),(2,0)')) AS RESULT;
result

t
(1 row)
```
- **isopen(path)**  
Description: An open path?  
Return type: Boolean  
Example:

```
gaussdb=# SELECT isopen(path '[(0,0),(1,1),(2,0)]') AS RESULT;
result

t
(1 row)
```

- **length(object)**

Description: Length calculation

Return type: double precision

Example:

```
gaussdb=# SELECT length(path '((-1,0),(1,0))') AS RESULT;
result

4
(1 row)
```

- **npoints(path)**

Description: Number of points in path

Return type: int

Example:

```
gaussdb=# SELECT npoints(path '[(0,0),(1,1),(2,0)]') AS RESULT;
result

3
(1 row)
```

- **npoints(polygon)**

Description: Number of points in polygon

Return type: int

Example:

```
gaussdb=# SELECT npoints(polygon '((1,1),(0,0))') AS RESULT;
result

2
(1 row)
```

- **pclose(path)**

Description: Converts path to closed.

Return type: path

Example:

```
gaussdb=# SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;
result

((0,0),(1,1),(2,0))
(1 row)
```

- **popen(path)**

Description: Converts path to open.

Return type: path

Example:

```
gaussdb=# SELECT popen(path '((0,0),(1,1),(2,0))') AS RESULT;
result

[(0,0),(1,1),(2,0)]
(1 row)
```

- **radius(circle)**

Description: Circle diameter calculation

Return type: double precision

Example:

```
gaussdb=# SELECT radius(circle '((0,0),2.0)') AS RESULT;
result

2
(1 row)
```

- width(box)

Description: Horizontal size of box

Return type: double precision

Example:

```
gaussdb=# SELECT width(box '((0,0),(1,1)')) AS RESULT;
result

1
(1 row)
```

## Geometric Type Conversion Functions

- box(circle)

Description: Circle to box

Return type: box

Example:

```
gaussdb=# SELECT box(circle '((0,0),2.0)') AS RESULT;
result

(1.41421356237309,1.41421356237309),(-1.41421356237309,-1.41421356237309)
(1 row)
```

- box(point, point)

Description: Points to box

Return type: box

Example:

```
gaussdb=# SELECT box(point '(0,0)', point '(1,1)') AS RESULT;
result

(1,1),(0,0)
(1 row)
```

- box(polygon)

Description: Polygon to box

Return type: box

Example:

```
gaussdb=# SELECT box(polygon '((0,0),(1,1),(2,0))') AS RESULT;
result

(2,1),(0,0)
(1 row)
```

- circle(box)

Description: Box to circle

Return type: circle

Example:

```
gaussdb=# SELECT circle(box '((0,0),(1,1)')) AS RESULT;
result
```

- ```
-----
<(0.5,0.5),0.707106781186548>
(1 row)
```
- circle(point, double precision)**
 Description: Center and radius to circle
 Return type: circle
 Example:

```
gaussdb=# SELECT circle(point '(0,0)', 2.0) AS RESULT;
result
-----
<(0,0),2>
(1 row)
```
 - circle(polygon)**
 Description: Polygon to circle
 Return type: circle
 Example:

```
gaussdb=# SELECT circle(polygon '((0,0),(1,1),(2,0))') AS RESULT;
result
-----
<(1,0.3333333333333333),0.924950591148529>
(1 row)
```
 - lseg(box)**
 Description: Box diagonal to line segment
 Return type: lseg
 Example:

```
gaussdb=# SELECT lseg(box '((-1,0),(1,0))') AS RESULT;
result
-----
[(1,0),(-1,0)]
(1 row)
```
 - lseg(point, point)**
 Description: Points to line segment
 Return type: lseg
 Example:

```
gaussdb=# SELECT lseg(point '(-1,0)', point '(1,0)') AS RESULT;
result
-----
[(-1,0),(1,0)]
(1 row)
```
 - slope(point, point)**
 Description: Calculates the slope of a straight line formed by two points.
 Return type: double
 Example:

```
gaussdb=# SELECT slope(point '(1,1)', point '(0,0)') AS RESULT;
result
-----
1
(1 row)
```
 - path(polygon)**
 Description: Polygon to path
 Return type: path

Example:

```
gaussdb=# SELECT path(polygon '((0,0),(1,1),(2,0)')) AS RESULT;  
result  
-----  
(0,0),(1,1),(2,0)  
(1 row)
```

- **point(double precision, double precision)**

Description: Points

Return type: point

Example:

```
gaussdb=# SELECT point(23.4, -44.5) AS RESULT;  
result  
-----  
(23.4,-44.5)  
(1 row)
```

- **point(box)**

Description: Center of box

Return type: point

Example:

```
gaussdb=# SELECT point(box '((-1,0),(1,0)')) AS RESULT;  
result  
-----  
(0,0)  
(1 row)
```

- **point(circle)**

Description: Center of circle

Return type: point

Example:

```
gaussdb=# SELECT point(circle '((0,0),2.0)') AS RESULT;  
result  
-----  
(0,0)  
(1 row)
```

- **point(lseg)**

Description: Center of line segment

Return type: point

Example:

```
gaussdb=# SELECT point(lseg '((-1,0),(1,0)')) AS RESULT;  
result  
-----  
(0,0)  
(1 row)
```

- **point(polygon)**

Description: Center of polygon

Return type: point

Example:

```
gaussdb=# SELECT point(polygon '((0,0),(1,1),(2,0)')) AS RESULT;  
result  
-----  
(1,0.3333333333333333)  
(1 row)
```

- **polygon(box)**
Description: Box to 4-point polygon
Return type: polygon

Example:

```
gaussdb=# SELECT polygon(box '((0,0),(1,1)')) AS RESULT;  
result  
-----  
((0,0),(0,1),(1,1),(1,0))  
(1 row)
```

- **polygon(circle)**
Description: Circle to 12-point polygon
Return type: polygon

Example:

```
gaussdb=# SELECT polygon(circle '((0,0),2.0)') AS RESULT;  
result  
-----  
-----  
((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),  
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),  
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),  
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))  
(1 row)
```

- **polygon(npts, circle)**
Description: Circle to npts-point polygon
Return type: polygon

Example:

```
gaussdb=# SELECT polygon(12, circle '((0,0),2.0)') AS RESULT;  
result  
-----  
-----  
((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),  
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),  
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),  
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))  
(1 row)
```

- **polygon(path)**
Description: Path to polygon
Return type: polygon

Example:

```
gaussdb=# SELECT polygon(path '((0,0),(1,1),(2,0)')) AS RESULT;  
result  
-----  
((0,0),(1,1),(2,0))  
(1 row)
```

7.5.11 Network Address Functions and Operators

cidr and inet Operators

The operators <<, <<=, >>, and >>= test for subnet inclusion. They consider only the network parts of the two addresses (ignoring any host part) and determine whether one network is identical to or a subnet of the other.

- <

Description: Is less than

Example:

```
gaussdb=# SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;
result
-----
t
(1 row)
```

- <=

Description: Is less than or equals

Example:

```
gaussdb=# SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;
result
-----
t
(1 row)
```

- =

Description: Equals

Example:

```
gaussdb=# SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;
result
-----
t
(1 row)
```

- >=

Description: Is greater than or equals

Example:

```
gaussdb=# SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;
result
-----
t
(1 row)
```

- >

Description: Is greater than

Example:

```
gaussdb=# SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;
result
-----
t
(1 row)
```

- <>

Description: Does not equal

Example:

```
gaussdb=# SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;
result
```

- ```

t
(1 row)
```

<<

Description: Is contained in

Example:

```
gaussdb=# SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;
result

t
(1 row)
```
- <<=
- >>
- >>=
- ~
- &
- |

Description: Performs an OR operation on each bit of the two network addresses.

Example:

```
gaussdb=# SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;
result

202.168.1.6
(1 row)
```

- +

Description: Addition

Example:

```
gaussdb=# SELECT inet '192.168.1.6' + 25 AS RESULT;
result

192.168.1.31
(1 row)
```

- -

Description: Subtraction

Example:

```
gaussdb=# SELECT inet '192.168.1.43' - 36 AS RESULT;
result

192.168.1.7
(1 row)
```

- -

Description: Subtraction

Example:

```
gaussdb=# SELECT inet '192.168.1.43' - inet '192.168.1.19' AS RESULT;
result

24
(1 row)
```

## cidr and inet Functions

The **abbrev**, **host**, and **text** functions are primarily intended to offer alternative display formats.

- abbrev(inet)

Description: Abbreviated display format as text

Return type: text

Example:

```
gaussdb=# SELECT abbrev(inet '10.1.0.0/16') AS RESULT;
result

10.1.0.0/16
(1 row)
```

- abbrev(cidr)

Description: Abbreviated display format as text

Return type: text

Example:

```
gaussdb=# SELECT abbrev(cidr '10.1.0.0/16') AS RESULT;
result
```

- ```
-----  
10.1/16  
(1 row)
```

 - **broadcast(inet)**
Description: Broadcast address for networks
Return type: inet
Example:

```
gaussdb=# SELECT broadcast('192.168.1.5/24') AS RESULT;  
result  
-----  
192.168.1.255/24  
(1 row)
```
 - **family(inet)**
Description: Extracts family of addresses, 4 for IPv4.
Return type: int
Example:

```
gaussdb=# SELECT family('127.0.0.1') AS RESULT;  
result  
-----  
4  
(1 row)
```
 - **host(inet)**
Description: Extracts IP addresses as text.
Return type: text
Example:

```
gaussdb=# SELECT host('192.168.1.5/24') AS RESULT;  
result  
-----  
192.168.1.5  
(1 row)
```
 - **hostmask(inet)**
Description: Constructs the host mask for a network.
Return type: inet
Example:

```
gaussdb=# SELECT hostmask('192.168.23.20/30') AS RESULT;  
result  
-----  
0.0.0.3  
(1 row)
```
 - **masklen(inet)**
Description: Extracts subnet mask length.
Return type: int
Example:

```
gaussdb=# SELECT masklen('192.168.1.5/24') AS RESULT;  
result  
-----  
24  
(1 row)
```
 - **netmask(inet)**
Description: Constructs the subnet mask for a network.
Return type: inet

Example:

```
gaussdb=# SELECT netmask('192.168.1.5/24') AS RESULT;
 result
-----
255.255.255.0
(1 row)
```

- `network(inet)`

Description: Extracts the network part of an address.

Return type: `cidr`

Example:

```
gaussdb=# SELECT network('192.168.1.5/24') AS RESULT;
 result
-----
192.168.1.0/24
(1 row)
```

- `set_masklen(inet, int)`

Description: Sets subnet mask length for the **inet** value.

Return type: `inet`

Example:

```
gaussdb=# SELECT set_masklen('192.168.1.5/24', 16) AS RESULT;
 result
-----
192.168.1.5/16
(1 row)
```

- `set_masklen(cidr, int)`

Description: Sets subnet mask length for the **cidr** value.

Return type: `cidr`

Example:

```
gaussdb=# SELECT set_masklen('192.168.1.0/24'::cidr, 16) AS RESULT;
 result
-----
192.168.0.0/16
(1 row)
```

- `text(inet)`

Description: Extracts IP addresses and subnet mask length as text.

Return type: `text`

Example:

```
gaussdb=# SELECT text(inet '192.168.1.5') AS RESULT;
 result
-----
192.168.1.5/32
(1 row)
```

Any **cidr** value can be cast to **inet** implicitly or explicitly; therefore, the functions shown above as operating on **inet** also work on **cidr** values. An **inet** value can be cast to **cidr**. After the conversion, any bits to the right of the subnet mask are silently zeroed to create a valid **cidr** value. In addition, you can cast a text string to **inet** or **cidr** using normal casting syntax. For example, **inet(expression)** or **colname::cidr**.

macaddr Functions

The function **trunc(macaddr)** returns a MAC address with the last 3 bytes set to zero.

trunc(macaddr)

Description: Sets last 3 bytes to zero.

Return type: macaddr

Example:

```
gaussdb=# SELECT trunc(macaddr '12:34:56:78:90:ab') AS RESULT;
result
-----
12:34:56:00:00:00
(1 row)
```

The macaddr type also supports the standard relational operators (such as > and <=) for lexicographical ordering, and the bitwise arithmetic operators (~, & and |) for NOT, AND and OR.

7.5.12 Text Search Functions and Operators

Text Search Operators

- @@

Description: Specifies whether the tsvector-typed words match the tsquery-typed words.

Example:

```
gaussdb=# SELECT to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat') AS RESULT;
result
-----
t
(1 row)
```

- @@@

Description: Synonym for @@

Example:

```
gaussdb=# SELECT to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat') AS RESULT;
result
-----
t
(1 row)
```

- ||

Description: Connects two tsvector-typed words.

Example:

```
gaussdb=# SELECT 'a:1 b:2'::tsvector || 'c:1 d:2 b:3'::tsvector AS RESULT;
result
-----
'a:1 'b':2,5 'c':3 'd':4
(1 row)
```

- &&

Description: Performs the AND operation on two tsquery-typed words.

Example:

```
gaussdb=# SELECT 'fat | rat'::tsquery && 'cat'::tsquery AS RESULT;
result
-----
( 'fat' | 'rat' ) & 'cat'
(1 row)
```

- ||
Description: Performs the OR operation on two tsquery-typed words.

Example:

```
gaussdb=# SELECT 'fat | rat'::tsquery || 'cat'::tsquery AS RESULT;
result
-----
( 'fat' | 'rat' ) | 'cat'
(1 row)
```

- !!
Description: Not a tsquery-typed word.

Example:

```
gaussdb=# SELECT !! 'cat'::tsquery AS RESULT;
result
-----
!'cat'
(1 row)
```

- @>
Description: Specifies whether a tsquery-typed word contains another tsquery-typed word.

Example:

```
gaussdb=# SELECT 'cat'::tsquery @> 'cat & rat'::tsquery AS RESULT;
result
-----
f
(1 row)
```

- <@
Description: Specifies whether a tsquery-typed word is contained in another tsquery-typed word.

Example:

```
gaussdb=# SELECT 'cat'::tsquery <@ 'cat & rat'::tsquery AS RESULT;
result
-----
t
(1 row)
```

In addition to the preceding operators, the ordinary B-tree comparison operators (including = and <) are defined for types tsvector and tsquery.

Text Search Functions

- get_current_ts_config()
Description: Obtains default text search configurations.
Return type: regconfig

Example:

```
gaussdb=# SELECT get_current_ts_config();
get_current_ts_config
-----
english
(1 row)
```

- length(tsvector)**
 Description: Specifies the number of tsvector-typed words.
 Return type: integer
 Example:

```
gaussdb=# SELECT length('fat:2,4 cat:3 rat:5A':tsvector);
length
-----
      3
(1 row)
```
- numnode(tsquery)**
 Description: Specifies the number of tsquery-typed words plus operators.
 Return type: integer
 Example:

```
gaussdb=# SELECT numnode('(fat & rat) | cat':tsquery);
numnode
-----
      5
(1 row)
```
- plainto_tsquery([config regconfig ,] query text)**
 Description: Generates tsquery-typed words without punctuations.
 Return type: tsquery
 Example:

```
gaussdb=# SELECT plainto_tsquery('english', 'The Fat Rats');
plainto_tsquery
-----
'fat' & 'rat'
(1 row)
```
- querytree(query tsquery)**
 Description: Obtains the indexable part of a tsquery-typed word.
 Return type: text
 Example:

```
gaussdb=# SELECT querytree('foo & ! bar':tsquery);
querytree
-----
'foo'
(1 row)
```
- setweight(tsvector, "char")**
 Description: Assigns weight to each element of tsvector.
 Return type: tsvector
 Example:

```
gaussdb=# SELECT setweight('fat:2,4 cat:3 rat:5B':tsvector, 'A');
setweight
-----
'cat':3A 'fat':2A,4A 'rat':5A
(1 row)
```
- strip(tsvector)**
 Description: Removes positions and weights from tsvector.
 Return type: tsvector
 Example:

```
gaussdb=# SELECT strip('fat:2,4 cat:3 rat:5A':tsvector);
strip
```

- ```

'cat' 'fat' 'rat'
(1 row)
```
- `to_tsquery([ config regconfig , ] query text)`  
 Description: Normalizes words and converts them to tsquery.  
 Return type: tsquery  
 Example:  

```
gaussdb=# SELECT to_tsquery('english', 'The & Fat & Rats');
to_tsquery

'fat' & 'rat'
(1 row)
```
  - `to_tsvector([ config regconfig , ] document text)`  
 Description: Reduces document text to tsvector.  
 Return type: tsvector  
 Example:  

```
gaussdb=# SELECT to_tsvector('english', 'The Fat Rats');
to_tsvector

'fat':2 'rat':3
(1 row)
```
  - `to_tsvector_for_batch([ config regconfig , ] document text)`  
 Description: Reduces document text to tsvector.  
 Return type: tsvector  
 Example:  

```
gaussdb=# SELECT to_tsvector_for_batch('english', 'The Fat Rats');
to_tsvector

'fat':2 'rat':3
(1 row)
```
  - `ts_headline([ config regconfig, ] document text, query tsquery [, options text ])`  
 Description: Highlights a query match.  
 Return type: text  
 Example:  

```
gaussdb=# SELECT ts_headline('x y z', 'z!::tsquery');
ts_headline

x y z
(1 row)
```
  - `ts_rank([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ])`  
 Description: Ranks document for query.  
 Return type: float4  
 Example:  

```
gaussdb=# SELECT ts_rank('hello world'::tsvector, 'world'::tsquery);
ts_rank

.0607927
(1 row)
```
  - `ts_rank_cd([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ])`

Description: Ranks document for query using cover density.

Return type: float4

Example:

```
gaussdb=# SELECT ts_rank_cd('hello world':tsvector, 'world':tsquery);
ts_rank_cd

.0
(1 row)
```

- `ts_rewrite(query tsquery, target tsquery, substitute tsquery)`

Description: Replaces tsquery-typed word.

Return type: tsquery

Example:

```
gaussdb=# SELECT ts_rewrite('a & b':tsquery, 'a':tsquery, 'foo|bar':tsquery);
ts_rewrite

'b' & ('foo' | 'bar')
(1 row)
```

- `ts_rewrite(query tsquery, select text)`

Description: Replaces tsquery data in the target with the result of a **SELECT** command.

Return type: tsquery

Example:

```
gaussdb=# SELECT ts_rewrite('world':tsquery, 'select "world":tsquery, "hello":tsquery');
ts_rewrite

'hello'
(1 row)
```

## Text Search Debugging Functions

- `ts_debug([ config regconfig, ] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])`

Description: Tests a configuration.

Return type: SETOF record

Example:

```
gaussdb=# SELECT ts_debug('english', 'The Brightest supernovaes');
ts_debug

(asciiword,"Word, all ASCII",The,{english_stem},english_stem,{})
(blank,"Space symbols","",{},{,})
(asciiword,"Word, all ASCII",Brightest,{english_stem},english_stem,{brightest})
(blank,"Space symbols","",{},{,})
(asciiword,"Word, all ASCII",supernovaes,{english_stem},english_stem,{supernova})
(5 rows)
```

- `ts_lexize(dict regdictionary, token text)`

Description: Tests a data dictionary.

Return type: text[]

Example:

```
gaussdb=# SELECT ts_lexize('english_stem', 'stars');
ts_lexize

{star}
(1 row)
```

- `ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)`

Description: Tests a parser.

Return type: SETOF record

Example:

```
gaussdb=# SELECT ts_parse('default', 'foo - bar');
ts_parse

(1,foo)
(12," ")
(12,"- ")
(1,bar)
(4 rows)
```

- `ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)`

Description: Tests a parser.

Return type: SETOF record

Example:

```
gaussdb=# SELECT ts_parse(3722, 'foo - bar');
ts_parse

(1,foo)
(12," ")
(12,"- ")
(1,bar)
(4 rows)
```

- `ts_token_type(parser_name text, OUT tokid integer, OUT alias text, OUT description text)`

Description: Obtains token types defined by a parser.

Return type: SETOF record

Example:

```
gaussdb=# SELECT ts_token_type('default');
ts_token_type

(1,asciword,"Word, all ASCII")
(2,word,"Word, all letters")
(3,numword,"Word, letters and digits")
(4,email,"Email address")
(5,url,URL)
(6,host,Host)
(7,sfloat,"Scientific notation")
(8,version,"Version number")
(9,hword_numpart,"Hyphenated word part, letters and digits")
(10,hword_part,"Hyphenated word part, all letters")
(11,hword_asciipart,"Hyphenated word part, all ASCII")
(12,blank,"Space symbols")
(13,tag,"XML tag")
(14,protocol,"Protocol head")
(15,numhword,"Hyphenated word, letters and digits")
(16,asciuhword,"Hyphenated word, all ASCII")
(17,hword,"Hyphenated word, all letters")
(18,url_path,"URL path")
(19,file,"File or path name")
(20,float,"Decimal notation")
(21,int,"Signed integer")
(22,uint,"Unsigned integer")
(23,entity,"XML entity")
(23 rows)
```

- `ts_token_type(parser_oid oid, OUT tokid integer, OUT alias text, OUT description text)`

Description: Obtains token types defined by a parser.

Return type: SETOF record

Example:

```
gaussdb=# SELECT ts_token_type(3722);
 ts_token_type

(1,asciiword,"Word, all ASCII")
(2,word,"Word, all letters")
(3,numword,"Word, letters and digits")
(4,email,"Email address")
(5,url,URL)
(6,host,Host)
(7,sfloat,"Scientific notation")
(8,version,"Version number")
(9,hword_numpart,"Hyphenated word part, letters and digits")
(10,hword_part,"Hyphenated word part, all letters")
(11,hword_asciipart,"Hyphenated word part, all ASCII")
(12,blank,"Space symbols")
(13,tag,"XML tag")
(14,protocol,"Protocol head")
(15,numhword,"Hyphenated word, letters and digits")
(16,asciihword,"Hyphenated word, all ASCII")
(17,hword,"Hyphenated word, all letters")
(18,url_path,"URL path")
(19,file,"File or path name")
(20,float,"Decimal notation")
(21,int,"Signed integer")
(22,uint,"Unsigned integer")
(23,entity,"XML entity")
(23 rows)
```

- `ts_stat(sqlquery text, [ weights text, ] OUT word text, OUT ndoc integer, OUT nentry integer)`

Description: Obtains statistics of a **tsvector** column.

Return type: SETOF record

Example:

```
gaussdb=# SELECT ts_stat('select "hello world"::tsvector');
 ts_stat

(world,1,1)
(hello,1,1)
(2 rows)
```

## 7.5.13 JSON/JSONB Functions and Operators

For details about the JSON/JSONB data type, see [JSON/JSONB Types](#). For details about operators, see [Table 7-38](#) and [Table 7-39](#).

**Table 7-38** JSON/JSONB common operators

| Operator | Left Operand Type     | Right Operand Type | Return Type | Description                                                                                         | Example                                                                                                  |
|----------|-----------------------|--------------------|-------------|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| ->       | Array-<br>json(b)     | int                | json(b)     | Obtains the <b>array-json</b> element. If the index does not exist, <b>NULL</b> is returned.        | SELECT '["a":"foo"],["b":"bar"],<br>{"c":"baz"}::json->2;<br>?column?<br>-----<br>{"c":"baz"}<br>(1 row) |
| ->       | object-<br>json(b)    | text               | json(b)     | Obtains the value by a key. If no record is found, <b>NULL</b> is returned.                         | SELECT '{"a": {"b":"foo"}}::json->'a';<br>?column?<br>-----<br>{"b":"foo"}<br>(1 row)                    |
| ->>      | Array-<br>json(b)     | int                | text        | Obtains the JSON array element. If the index does not exist, <b>NULL</b> is returned.               | SELECT '[1,2,3]::json->>2;<br>?column?<br>-----<br>3<br>(1 row)                                          |
| ->>      | object-<br>json(b)    | text               | text        | Obtains the value by a key. If no record is found, <b>NULL</b> is returned.                         | SELECT '{"a":1,"b":2}::json->>'b';<br>?column?<br>-----<br>2<br>(1 row)                                  |
| #>       | container-<br>json(b) | text[ ]            | json(b)     | Obtains the JSON object in the specified path. If the path does not exist, <b>NULL</b> is returned. | SELECT '{"a": {"b":{"c": "foo"}}}::json<br>#>'a,b';<br>?column?<br>-----<br>{"c": "foo"}<br>(1 row)      |
| #>>      | container-<br>json(b) | text[ ]            | text        | Obtains the JSON object in the specified path. If the path does not exist, <b>NULL</b> is returned. | SELECT '{"a":[1,2,3],"b":[4,5,6]}::json<br>#>>'a,2';<br>?column?<br>-----<br>3<br>(1 row)                |

 **CAUTION**

For the #> and #>> operators, if no data can be found in the specified path, no error is reported and a **NULL** value is returned.

**Table 7-39** Additional JSONB support for operators

| Operators | Right Operand Type | Description                                                                                                           | Example                                           |
|-----------|--------------------|-----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| @>        | jsonb              | Specifies whether the top layer of the JSON on the left contains all items of the top layer of the JSON on the right. | '{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb       |
| <@        | jsonb              | Specifies whether all items in the JSON file on the left exist at the top layer of the JSON file on the right.        | '{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb       |
| ?         | text               | Specifies whether the string of the key or element exists at the top layer of the JSON value.                         | '{"a":1, "b":2}'::jsonb ? 'b'                     |
| ?         | text[]             | Specifies whether any of these array strings exists as top-layer keys.                                                | '{"a":1, "b":2, "c":3}'::jsonb ?  array['b', 'c'] |
| ?&        | text[]             | Specifies whether all these array strings exist as top-layer keys.                                                    | '["a", "b"]'::jsonb ? & array['a', 'b']           |
| =         | jsonb              | Determines the size between two <b>JSONB</b> files, which is the same as the <b>jsonb_eq</b> function.                | /                                                 |
| <>        | jsonb              | Determines the size between two <b>JSONB</b> files, which is the same as the <b>jsonb_ne</b> function.                | /                                                 |

| Operators | Right Operand Type | Description                                                                                            | Example |
|-----------|--------------------|--------------------------------------------------------------------------------------------------------|---------|
| <         | jsonb              | Determines the size between two <b>JSONB</b> files, which is the same as the <b>jsonb_lt</b> function. | /       |
| >         | jsonb              | Determines the size between two <b>JSONB</b> files, which is the same as the <b>jsonb_gt</b> function. | /       |
| <=        | jsonb              | Determines the size between two <b>JSONB</b> files, which is the same as the <b>jsonb_le</b> function. | /       |
| >=        | jsonb              | Determines the size between two <b>JSONB</b> files, which is the same as the <b>jsonb_ge</b> function. | /       |

## Functions Supported by JSON/JSONB

- array\_to\_json(anyarray [, pretty\_bool])  
Description: Returns the array as JSON. It combines a multi-dimensional array into a JSON array. Line feeds will be added between one-dimensional elements if **pretty\_bool** is **true**.

Return type: json

Example:

```
gaussdb=# SELECT array_to_json('{{1,5},{99,100}}':int[]);
array_to_json

[[1,5],[99,100]]
(1 row)
```

- json\_array\_element(array-json, integer), jsonb\_array\_element(array-jsonb, integer)

Description: Same as the operator `->`, which returns the element with the specified index in the array.

Return type: json, jsonb

Example:

```
gaussdb=# SELECT json_array_element('[1,true,[1,[2,3]],null]',2);
```

```
json_array_element

[1,[2,3]]
(1 row)
```

- `json_array_element_text(array-json, integer)`, `jsonb_array_element_text(array-jsonb, integer)`

Description: Same as the operator ``->>``, which returns the element with the specified index in the array.

Return type: text, text

Example:

```
gaussdb=# SELECT json_array_element_text('[1,true,[1,[2,3]],null]',2);
json_array_element_text

[1,[2,3]]
(1 row)
```

- `json_object_field(object-json, text)`, `jsonb_object_field(object-jsonb, text)`

Description: Same as the operator ``->``, which returns the value of a specified key in an object.

Return type: json, json

Example:

```
gaussdb=# SELECT json_object_field('{\"a\": {\"b\":\"foo\"}}','a');
json_object_field

{\"b\":\"foo\"}
(1 row)
```

- `json_object_field_text(object-json, text)`, `jsonb_object_field_text(object-jsonb, text)`

Description: Same as the operator ``->``, which returns the value of a specified key in an object.

Return type: text, text

Example:

```
gaussdb=# SELECT json_object_field_text('{\"a\": {\"b\":\"foo\"}}','a');
json_object_field_text

{\"b\":\"foo\"}
(1 row)
```

- `json_extract_path(json, VARIADIC text[])`, `jsonb_extract_path((jsonb, VARIADIC text[])`

Description: Equivalent to the operator ``#>`` searches for JSON based on the path specified by `$2` and returns the result.

Return type: json, jsonb

Example:

```
gaussdb=# SELECT json_extract_path('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99,\"f6\":\"stringy\"}}', 'f4','f6');
json_extract_path

\"stringy\"
(1 row)
```

- `json_extract_path_op(json, text[])`, `jsonb_extract_path_op(jsonb, text[])`

Description: Same as the operator ``#>`` and searches for JSON based on the path specified by `$2` and returns the result.

Return type: json, jsonb

Example:

```
gaussdb=# SELECT json_extract_path_op('{ "f2": {"f3": 1}, "f4": {"f5": 99, "f6": "stringy"} }', ARRAY['f4', 'f6']);
json_extract_path_op

"stringy"
(1 row)
```

- `json_extract_path_text(json, VARIADIC text[]), jsonb_extract_path_text((jsonb, VARIADIC text[])`

Description: Same as the operator ``#>`` and searches for JSON based on the path specified by `$2` and returns the result.

Return type: text, text

Example:

```
gaussdb=# SELECT json_extract_path_text('{ "f2": {"f3": 1}, "f4": {"f5": 99, "f6": "stringy"} }', 'f4', 'f6');
json_extract_path_text

"stringy"
(1 row)
```

- `json_extract_path_text_op(json, text[]), jsonb_extract_path_text_op(jsonb, text[])`

Description: Same as the operator ``#>>`` and searches for JSON based on the path specified by `$2` and returns the result.

Return type: text, text

Example:

```
gaussdb=# SELECT json_extract_path_text_op('{ "f2": {"f3": 1}, "f4": {"f5": 99, "f6": "stringy"} }', ARRAY['f4', 'f6']);
json_extract_path_text_op

"stringy"
(1 row)
```

- `json_array_elements(array-json), jsonb_array_elements(array-jsonb)`

Description: Splits an array. Each element returns a row.

Return type: json, jsonb

Example:

```
gaussdb=# SELECT json_array_elements('[1,true,[1,[2,3]],null]');
json_array_elements

1
true
[1,[2,3]]
null
(4 rows)
```

- `json_array_elements_text(array-json), jsonb_array_elements_text(array-jsonb)`

Description: Splits an array. Each element returns a row.

Return type: text, text

Example:

```
gaussdb=# SELECT * FROM json_array_elements_text('[1,true,[1,[2,3]],null]');
value

1
true
[1,[2,3]]
(4 rows)
```

- `json_array_length(array-json), jsonb_array_length(array-jsonb)`

Description: Returns the array length.

Return type: integer

Example:

```
gaussdb=# SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4,null]');
 json_array_length

 6
(1 row)
```

- `json_each(object-json), jsonb_each(object-jsonb)`

Description: Splits each key-value pair of an object into one row and two columns.

Return type: `setof(key text, value json), setof(key text, value jsonb)`

Example:

```
gaussdb=# SELECT * FROM json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
 key | value
-----+-----
 f1 | [1,2,3]
 f2 | {"f3":1}
 f4 | null
(3 rows)
```

- `json_each_text(object-json), jsonb_each_text(object-jsonb)`

Description: Splits each key-value pair of an object into one row and two columns.

Return type: `setof(key text, value text), setof(key text, value text)`

Example:

```
gaussdb=# SELECT * FROM json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
 key | value
-----+-----
 f1 | [1,2,3]
 f2 | {"f3":1}
 f4 |
(3 rows)
```

- `json_object_keys(object-json), jsonb_object_keys(object-jsonb)`

Description: Returns all keys at the top layer of the object.

Return type: `SETOF text`

Example:

```
gaussdb=# SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a"}, "f4":"b"}, {"f1":"abcd"}');
 json_object_keys

 f1
 f2
 f1
(3 rows)
```

- **JSONB deduplication operations:**

```
gaussdb=# SELECT jsonb_object_keys('{"f1":"abc","f2":{"f3":"a"}, "f4":"b"}, {"f1":"abcd"}');
 jsonb_object_keys

 f1
 f2
(2 rows)
```

- `json_typeof(json), jsonb_typeof(jsonb)`

Description: Checks the JSON type.

Return type: `text, text`

Example:

```
gaussdb=# SELECT value, json_typeof(value) FROM (values (json '123.4'), (json "'foo'"), (json 'true'),
(json 'null'), (json '[1, 2, 3]'), (json '{"x":"foo", "y":123}'), (NULL::json)) AS data(value);
 value | json_typeof
-----+-----
 123.4 | number
```

```

"foo" | string
true | boolean
null | null
[1, 2, 3] | array
{"x":"foo", "y":123} | object
|
(7 rows)

```

- `json_build_array( [VARIADIC "any"] )`

Description: Constructs a JSON array from a variable parameter list.

Return type: array-json

Example:

```

gaussdb=# SELECT json_build_array('a',1,'b',1.2,'c',true,'d',null,'e',json '{"x": 3, "y": [1,2,3]}');
 json_build_array

["a", 1, "b", 1.2, "c", true, "d", null, "e", {"x": 3, "y": [1,2,3]}, ""]
(1 row)

```

- `json_build_object( [VARIADIC "any"] )`

Description: Constructs a JSON object from a variable parameter list. The number of input parameters must be an even number. Every two input parameters form a key-value pair. Note that the value of a key cannot be null.

Return type: object-json

Example:

```

gaussdb=# SELECT json_build_object(1,2);
 json_build_object

{"1" : 2}
(1 row)

```

- `json_object(text[]), json_object(text[], text[])`

Description: Constructs an object-json from a text array. This is an overloaded function. When the input parameter is a text array, the array length must be an even number, and members are considered as alternate key-value pairs. When two text arrays are used, the first array is considered as a key, and the second array a value. The lengths of the two arrays must be the same. Note that the value of a key cannot be null.

Return type: object-json

Example:

```

gaussdb=# SELECT json_object('{a,1,b,2,3,NULL,"d e f","a b c"}');
 json_object

{"a" : "1", "b" : "2", "3" : null, "d e f" : "a b c"}
(1 row)
gaussdb=# SELECT json_object('{a,b,"a b c"}', '{a,1,1}');
 json_object

{"a" : "a", "b" : "1", "a b c" : "1"}
(1 row)

```

- `json_agg(any)`

Description: Aggregates values into a JSON array.

Return type: array-json

Example:

```

gaussdb=# SELECT * FROM classes;
 name | score
-----+-----
 A | 2
 A | 3
 D | 5

```

```
D |
(4 rows)
gaussdb=# SELECT name, json_agg(score) score FROM classes GROUP BY name ORDER BY name;
name | score
-----+-----
A | [2, 3]
D | [5, null]
| [null]
(3 rows)
```

- `json_object_agg(any, any)`

Description: Aggregates values into a JSON object.

Return type: object-json

Example:

```
gaussdb=# SELECT * FROM classes;
name | score
-----+-----
A | 2
A | 3
D | 5
D |
(4 rows)
```

```
gaussdb=# SELECT json_object_agg(name, score) FROM classes GROUP BY name ORDER BY name;
json_object_agg

{ "A" : 2, "A" : 3 }
{ "D" : 5, "D" : null }
(2 rows)
```

- `- jsonb_contained(jsonb, jsonb)`

Description: Same as the operator `<@`, determines whether all elements in `$1` exist at the top layer of `$2`.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_contained('[1,2,3]', '[1,2,3,4]');
jsonb_contained

t
(1 row)
```

- `- jsonb_contains(jsonb, jsonb)`

Description: Same as the operator `@>`, checks whether all top-layer elements in `$1` are contained in `$2`.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_contains('[1,2,3,4]', '[1,2,3]');
jsonb_contains

t
(1 row)
```

- `- jsonb_exists(jsonb, text)`

Description: Same as the operator `?>`, determines whether all elements in the string array `$2` exist at the top layer of `$1` in the form of **key\elem\scalar**.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_exists('["1",2,3]', '1');
jsonb_exists

t
(1 row)
```

- jsonb\_exists\_all(jsonb, text[])

Description: Same as the operator `?&`, checks whether all elements in the string array \$2 exist at the top layer of \$1 in the form of **key\elem\scalar**.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_exists_all('["1","2",3]', '{1, 2}');
 jsonb_exists_all

 t
(1 row)
```
- jsonb\_exists\_any(jsonb, text[])

Description: Same as the operator `?|`, checks whether all elements in the string array \$2 exist at the top layer of \$1 in the form of **key\elem\scalar**.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_exists_any('["1","2",3]', '{1, 2, 4}');
 jsonb_exists_any

 t
(1 row)
```
- jsonb\_cmp(jsonb, jsonb)

Description: Compares values. A positive value indicates greater than, a negative value indicates less than, and **0** indicates equal.

Return type: integer

Example:

```
gaussdb=# SELECT jsonb_cmp('["a", "b"]', '{"a":1, "b":2}');
 jsonb_cmp

 -1
(1 row)
```
- jsonb\_eq(jsonb, jsonb)

Description: Same as the operator `=`, compares two values.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_eq('["a", "b"]', '{"a":1, "b":2}');
 jsonb_eq

 f
(1 row)
```
- jsonb\_ne(jsonb, jsonb)

Description: Same as the operator `<>`, compares two values.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_ne('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ne

 t
(1 row)
```
- jsonb\_gt(jsonb, jsonb)

Description: Same as the operator `>`, compares two values.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_gt('["a", "b"]', '{"a":1, "b":2}');
 jsonb_gt

 f
(1 row)
```

- - jsonb\_ge(jsonb, jsonb)

Description: Same as the operator `>=`, compares two values.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_ge('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ge

 f
(1 row)
```

- - jsonb\_lt(jsonb, jsonb)

Description: Same as the operator `<`, compares two values.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_lt('["a", "b"]', '{"a":1, "b":2}');
 jsonb_lt

 t
(1 row)
```

- - jsonb\_le(jsonb, jsonb)

Description: Same as the operator `<=`, compares two values.

Return type: Boolean

Example:

```
gaussdb=# SELECT jsonb_le('["a", "b"]', '{"a":1, "b":2}');
 jsonb_le

 t
(1 row)
```

- - to\_json(anyelement)

Description: Converts parameters to `json`.

Return type: json

Example:

```
gaussdb=# SELECT to_json('{1,5}::text[]');
 to_json

 ["1","5"]
(1 row)
```

- - jsonb\_hash(jsonb)

Description: Performs the hash operation on JSONB.

Return type: integer

Example:

```
gaussdb=# SELECT jsonb_hash('[1,2,3]');
 jsonb_hash

 -559968547
(1 row)
```

- Other functions

```
json_agg_transfn
json_agg_finalfn
```

```
json_object_agg_transfn
json_object_agg_finalfn
```

## 7.5.14 HLL Functions and Operators

### Hash Functions

- `hll_hash_boolean(bool)`

Description: Hashes data of the bool type.

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_boolean(FALSE);
hll_hash_boolean

-5451962507482445012
(1 row)
```

- `hll_hash_boolean(bool, int32)`

Description: Configures a hash seed (that is, change the hash policy) and hashes data of the bool type.

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_boolean(FALSE, 10);
hll_hash_boolean

-1169037589280886076
(1 row)
```

- `hll_hash_smallint(smallint)`

Description: Hashes data of the smallint type.

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_smallint(100::smallint);
hll_hash_smallint

962727970174027904
(1 row)
```

#### NOTE

If parameters with the same numeric value are hashed using different data types, the data will differ, because hash functions select different calculation policies for each type.

- `hll_hash_smallint(smallint, int32)`

Description: Configures a hash seed (that is, change the hash policy) and hashes data of the smallint type.

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_smallint(100::smallint, 10);
hll_hash_smallint

-9056177146160443041
(1 row)
```

- `hll_hash_integer(integer)`

Description: Hashes data of the integer type.

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_integer(0);
 hll_hash_integer

5156626420896634997
(1 row)
```

- `hll_hash_integer(integer, int32)`

Description: Hashes data of the integer type and configures a hash seed (that is, change the hash policy).

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_integer(0, 10);
 hll_hash_integer

-5035020264353794276
(1 row)
```

- `hll_hash_bigint(bigint)`

Description: Hashes data of the bigint type.

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_bigint(100::bigint);
 hll_hash_bigint

-2401963681423227794
(1 row)
```

- `hll_hash_bigint(bigint, int32)`

Description: Hashes data of the bigint type and configures a hash seed (that is, change the hash policy).

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_bigint(100::bigint, 10);
 hll_hash_bigint

-2305749404374433531
(1 row)
```

- `hll_hash_bytea(bytea)`

Description: Hashes data of the bytea type.

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_bytea(E'\x');
 hll_hash_bytea

0
(1 row)
```

- `hll_hash_bytea(bytea, int32)`

Description: Hashes data of the bytea type and configures a hash seed (that is, change the hash policy).

Return type: `hll_hashval`

Example:

```
gaussdb=# SELECT hll_hash_bytea(E'\x', 10);
 hll_hash_bytea

```

```
7233188113542599437
(1 row)
```

- **hll\_hash\_text(text)**

Description: Hashes data of the text type.

Return type: hll\_hashval

Example:

```
gaussdb=# SELECT hll_hash_text('AB');
 hll_hash_text

-5666002586880275174
(1 row)
```

- **hll\_hash\_text(text, int32)**

Description: Hashes data of the text type and configures a hash seed (that is, change the hash policy).

Return type: hll\_hashval

Example:

```
gaussdb=# SELECT hll_hash_text('AB', 10);
 hll_hash_text

-2215507121143724132
(1 row)
```

- **hll\_hash\_any(anytype)**

Description: Hashes data of any type.

Return type: hll\_hashval

Example:

```
gaussdb=# SELECT hll_hash_any(1);
 hll_hash_any

-1316670585935156930
(1 row)

gaussdb=# SELECT hll_hash_any('08:00:2b:01:02:03::macaddr');
 hll_hash_any

-3719950434455589360
(1 row)
```

- **hll\_hash\_any(anytype, int32)**

Description: Hashes data of any type and configures a hash seed (that is, change the hash policy).

Return type: hll\_hashval

Example:

```
gaussdb=# SELECT hll_hash_any(1, 10);
 hll_hash_any

7048553517657992351
(1 row)
```

- **hll\_hashval\_eq(hll\_hashval, hll\_hashval)**

Description: Compares two pieces of data of the hll\_hashval type to check whether they are the same.

Return type: Boolean

Example:

```
gaussdb=# SELECT hll_hashval_eq(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_eq
```

```

t
(1 row)
```

- `hll_hashval_ne(hll_hashval, hll_hashval)`

Description: Compares two pieces of data of the `hll_hashval` type to check whether they are different.

Return type: Boolean

Example:

```
gaussdb=# SELECT hll_hashval_ne(hll_hash_integer(1), hll_hash_integer(1));
hll_hashval_ne
```

```

f
(1 row)
```

## HLL Functions

There are three HLL modes: explicit, sparse, and full. When the data size is small, the explicit mode is used. In this mode, distinct values are calculated without errors. As the number of distinct values increases, the HLL mode is switched to the sparse and full modes in sequence. The two modes have no difference in the calculation result, but vary in the calculation efficiency of HLL functions and the storage space of HLL objects. The following functions can be used to view some HLL parameters:

- `hll_print(hll)`

Description: Prints some debugging parameters of an HLL.

Example:

```
gaussdb=# SELECT hll_print(hll_empty());
hll_print
```

```

type=1(HLL_EMPTY), log2m=14, log2explicit=10, log2sparse=12, duplicatecheck=0
(1 row)
```

- `hll_type(hll)`

Description: Checks the type of the current HLL. The return values are described as follows: **0** indicates **HLL\_UNINIT**, an HLL object that is not initialized. **1** indicates **HLL\_EMPTY**, an empty HLL object. **2** indicates **HLL\_EXPLICIT**, an HLL object in explicit mode. **3** indicates **HLL\_SPARSE**, an HLL object in sparse mode. **4** indicates **HLL\_FULL**, an HLL object in full mode. **5** indicates **HLL\_UNDEFINED**, an invalid HLL object.

Example:

```
gaussdb=# SELECT hll_type(hll_empty());
hll_type
```

```

1
(1 row)
```

- `hll_log2m(hll)`

Description: Checks the value of **log2m** in the current HLL data structure. **log2m** is the logarithm of the number of buckets. This value affects the error rate of calculating distinct values by HLL. The error rate =  $\pm 1.04/\sqrt{2^{\log 2m}}$ . If the value of **log2m** ranges from 10 to 16, HLL sets the number of buckets to  $2^{\log 2m}$ . When the value of **log2explicit** is explicitly set to **-1**, the built-in default value is used.

Example:

```
gaussdb=# SELECT hll_log2m(hll_empty());
hll_log2m

 14
(1 row)

gaussdb=# SELECT hll_log2m(hll_empty(10));
hll_log2m

 10
(1 row)

gaussdb=# SELECT hll_log2m(hll_empty(-1));
hll_log2m

 14
(1 row)
```

- **hll\_log2explicit(hll)**

Description: Queries the value of **log2explicit** in the current HLL data structure. Generally, the HLL changes from the explicit mode to the sparse mode and then to the full mode. This process is called the promotion hierarchy policy. You can change the value of **log2explicit** to change the policy. For example, if the value of **log2explicit** is **0**, the HLL will skip the explicit mode and directly enter the sparse mode. When the value of **log2explicit** is explicitly set to a value ranging from 1 to 12, the HLL will switch to the sparse mode when the length of the data segment exceeds  $2^{\text{log2explicit}}$ . When the value of **log2explicit** is explicitly set to **-1**, the built-in default value is used.

Example:

```
gaussdb=# SELECT hll_log2explicit(hll_empty());
hll_log2explicit

 10
(1 row)

gaussdb=# SELECT hll_log2explicit(hll_empty(12, 8));
hll_log2explicit

 8
(1 row)

gaussdb=# SELECT hll_log2explicit(hll_empty(12, -1));
hll_log2explicit

 10
(1 row)
```

- **hll\_log2sparse(hll)**

Description: Queries the value of **log2sparse** in the current HLL data structure. Generally, the HLL changes from the explicit mode to the sparse mode and then to the full mode. This process is called the promotion hierarchy policy. You can adjust the value of **log2sparse** to change the policy. For example, if the value of **log2sparse** is **0**, the system skips the sparse mode and directly enters the full mode. If the value of **log2sparse** is explicitly set to a value ranging from 1 to 14, the HLL will switch to the full mode when the length of the data segment exceeds  $2^{\text{log2sparse}}$ . When the value of **log2sparse** is explicitly set to **-1**, the built-in default value is used.

Example:

```
gaussdb=# SELECT hll_log2sparse(hll_empty());
hll_log2sparse
```







Example:

```
gaussdb=# SELECT hll_cardinality(hll_empty() || hll_hash_integer(1));
hll_cardinality

1
(1 row)
```

- hll\_union(hll, hll)

Description: Performs the UNION operation on two HLL data structures to obtain one HLL.

Return type: HLL

Example:

```
gaussdb=# SELECT hll_union(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
hll_union

\x484c4c10002000002b0900000000000000004000000000000000b3ccc49320cca1ae3e2921ff133fba
ed00
(1 row)
```

## Aggregate Functions

- hll\_add\_agg(hll\_hashval)

Description: Groups hashed data into HLL.

Return type: HLL

Example:

```
-- Prepare data:
gaussdb=# CREATE TABLE t_id(id int);
gaussdb=# INSERT INTO t_id VALUES(generate_series(1,500));
gaussdb=# CREATE TABLE t_data(a int, c text);
gaussdb=# INSERT INTO t_data SELECT mod(id,2), id FROM t_id;

-- Create a table and specify an HLL column:
gaussdb=# CREATE TABLE t_a_c_hll(a int, c hll);

-- Use GROUP BY on column a to group data, and insert the data to the HLL.
gaussdb=# INSERT INTO t_a_c_hll SELECT a, hll_add_agg(hll_hash_text(c)) FROM t_data GROUP BY a;

-- Calculate the number of distinct values for each group in the HLL:
gaussdb=# SELECT a, #c AS cardinality FROM t_a_c_hll ORDER BY a;
a | cardinality
--+-----
0 | 247.862354346299
1 | 250.908710610377
(2 rows)
```

- hll\_add\_agg(hll\_hashval, int32 log2m)

Description: Groups hashed data into HLL and specifies the **log2m** parameter. The value ranges from 10 to 16. If the input is **-1** or **NULL**, the built-in default value is used.

Return type: HLL

Example:

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), 12)) FROM t_data;
hll_cardinality

497.965240179228
(1 row)
```

- `hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit)`  
Description: Groups hashed data into HLL and specifies the **log2m** and **log2explicit** parameters in sequence. The value of **log2explicit** ranges from 0 to 12. The value **0** indicates that the explicit mode is skipped. This parameter is used to set the threshold of the explicit mode. When the length of the data segment reaches  $2^{\text{log2explicit}}$ , the mode is switched to the sparse or full mode. If the input is **-1** or **NULL**, the built-in default value of **log2explicit** is used.  
Return type: HLL  
Example:

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1)) FROM t_data;
hll_cardinality

498.496062953313
(1 row)
```
- `hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse)`  
Description: Groups hashed data into HLL and sets the **log2m**, **log2explicit**, and **log2sparse** parameters in sequence. The value of **log2sparse** ranges from 0 to 14. The value **0** indicates that the sparse mode is skipped. This parameter is used to set the threshold of the sparse mode. When the length of the data segment reaches  $2^{\text{log2sparse}}$ , the mode is switched to the full mode. If the input is **-1** or **NULL**, the built-in default value of **log2sparse** is used.  
Return type: HLL  
Example:

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10)) FROM t_data;
hll_cardinality

498.496062953313
(1 row)
```
- `hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)`  
Description: Groups hashed data into HLL and specifies the **log2m**, **log2explicit**, **log2sparse**, and **duplicatecheck** parameters. The value of **duplicatecheck** can be **0** or **1**, indicating whether to enable this mode. By default, this mode is disabled. If the input is **-1** or **NULL**, the built-in default value of **duplicatecheck** is used.  
Return type: HLL  
Example:

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10, -1)) FROM t_data;
hll_cardinality

498.496062953313
(1 row)
```
- `hll_union_agg(hll)`  
Description: Performs the UNION operation on multiple pieces of data of the HLL type to obtain one HLL.  
Return type: HLL  
Example:

```
-- Perform the UNION operation on data of the HLL type in each group to obtain one HLL, and
calculate the number of distinct values:
gaussdb=# SELECT #hll_union_agg(c) AS cardinality FROM t_a_c_hll;
cardinality

```

```
498.496062953313
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE t_id;
gaussdb=# DROP TABLE t_data;
gaussdb=# DROP TABLE t_a_c_hll;
```

 **NOTE**

To perform the UNION operation on data in multiple HLLs, ensure that the HLLs have the same precision. Otherwise, UNION cannot be performed. This constraint also applies to the `hll_union(hll, hll)` function.

## Obsolete Functions

Some old HLL functions are discarded due to version upgrade. You can replace them with similar functions.

- hll\_schema\_version(hll)**

Description: Checks the schema version in the current HLL. In earlier versions, the schema version is fixed at **1**, which is used to verify the header of the HLL field. After refactoring, the HLL field is added to the header for verification. The schema version is no longer used.
- hll\_regwidth(hll)**

Description: Queries the bucket size in the HLL data structure. In earlier versions, the value of **regwidth** ranges from 1 to 5, which has a large error and limits the upper limit of the cardinality estimation. After refactoring, the value of **regwidth** is fixed at **6** and the variable is not used.
- hll\_expthresh(hll)**

Description: Obtains the value of **expthresh** in the current HLL. The `hll_log2explicit(hll)` function is used to replace similar functions.
- hll\_sparseon(hll)**

Description: Specifies whether the sparse mode is enabled. Use `hll_log2sparse(hll)` to replace similar functions. The value **0** indicates that the sparse mode is disabled.

## Built-In Functions

HLL has a series of built-in functions for internal data processing. Generally, users do not need to know how to use these functions. For details, see [Table 7-40](#).

**Table 7-40** Built-in functions

| Function                  | Description                                                 |
|---------------------------|-------------------------------------------------------------|
| <code>hll_in</code>       | Receives HLL data in string format.                         |
| <code>hll_out</code>      | Sends HLL data in string format.                            |
| <code>hll_recv</code>     | Receives HLL data in bytea format.                          |
| <code>hll_send</code>     | Sends HLL data in bytea format.                             |
| <code>hll_trans_in</code> | Receives <code>hll_trans_type</code> data in string format. |

| Function          | Description                                                                                                                                                               |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hll_trans_out     | Sends hll_trans_type data in string format.                                                                                                                               |
| hll_trans_recv    | Receives hll_trans_type data in bytea format.                                                                                                                             |
| hll_trans_send    | Sends hll_trans_type data in bytea format.                                                                                                                                |
| hll_typmod_in     | Receives typmod data.                                                                                                                                                     |
| hll_typmod_out    | Sends typmod data.                                                                                                                                                        |
| hll_hashval_in    | Receives hll_hashval data.                                                                                                                                                |
| hll_hashval_out   | Sends hll_hashval data.                                                                                                                                                   |
| hll_add_trans0    | Works similar to hll_add. No input parameter is specified during initialization. It is usually used in the first phase of DNs in distributed aggregation operations.      |
| hll_add_trans1    | Works similar to hll_add. An input parameter is specified during initialization. It is usually used in the first phase of DNs in distributed aggregation operations.      |
| hll_add_trans2    | Works similar to hll_add. Two input parameters are specified during initialization. It is usually used in the first phase of DNs in distributed aggregation operations.   |
| hll_add_trans3    | Works similar to hll_add. Three input parameters are specified during initialization. It is usually used in the first phase of DNs in distributed aggregation operations. |
| hll_add_trans4    | Works similar to hll_add. Four input parameters are specified during initialization. It is usually used in the first phase of DNs in distributed aggregation operations.  |
| hll_union_trans   | Works similar to hll_union and is used on the first phase of DNs in distributed aggregation operations.                                                                   |
| hll_union_collect | Works similar to hll_union and is used on the second phase of CNs in distributed aggregation operations to summarize the results of each DN.                              |
| hll_pack          | Is used on the third phase of CNs in distributed aggregation operations to convert a user-defined type hll_trans_type to the HLL type.                                    |
| hll               | Converts an HLL type to another HLL type. Input parameters can be specified.                                                                                              |
| hll_hashval       | Converts the bigint type to the hll_hashval type.                                                                                                                         |
| hll_hashval_int4  | Converts the int4 type to the hll_hashval type.                                                                                                                           |

## Operators

- =**

Description: Compares the values of the HLL or hll\_hashval type to check whether they are the same.

Return type: Boolean

Example:

```
--hll
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) = (hll_empty() || hll_hash_integer(1));
column

t
(1 row)

--hll_hashval
gaussdb=# SELECT hll_hash_integer(1) = hll_hash_integer(1);
?column?

t
(1 row)
```
- <> or !=**

Description: Compares the values of the HLL or hll\_hashval type to check whether they are different.

Return type: Boolean

Example:

```
--hll
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) <> (hll_empty() || hll_hash_integer(2));
?column?

t
(1 row)

--hll_hashval
gaussdb=# SELECT hll_hash_integer(1) <> hll_hash_integer(2);
?column?

t
(1 row)
```
- ||**

Description: Represents the functions of hll\_add, hll\_union, and hll\_add\_rev.

Return type: HLL

Example:

```
--hll_add
gaussdb=# SELECT hll_empty() || hll_hash_integer(1);
?column?

\x484c4c08000002002b09000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_add_rev
gaussdb=# SELECT hll_hash_integer(1) || hll_empty();
?column?

\x484c4c08000002002b09000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_union
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) || (hll_empty() || hll_hash_integer(2));
?column?
```

```

\x484c4c1000200002b0900000000000000040000000000000000b3ccc49320cca1ae3e2921ff133fba
ed00
(1 row)
```

- **#**  
Description: Calculates the number of distinct values of an HLL. It works the same as the `hll_cardinality` function.

Return type: int

Example:

```
gaussdb=# SELECT #(hll_empty() || hll_hash_integer(1));
?column?

 1
(1 row)
```

## 7.5.15 SEQUENCE Functions

The sequence functions provide a simple method to ensure security of multiple users for users to obtain sequence values from sequence objects.

- `nextval(regclass)`

Description: Specifies an increasing sequence and returns a new value.

### NOTE

- To avoid blocking of concurrent transactions that obtain numbers from the same sequence, a `nextval` operation is never rolled back; that is, once a value is fetched, it is considered used, even if the transaction that did the `nextval` later aborts. This means that aborted transactions may leave unused "holes" in the sequence of assigned values. Therefore, sequences in GaussDB cannot be used to obtain sequence without gaps.
- If the `nextval` function is pushed to DN, each DN will automatically connect to the GTM and requests the next value. For example, in the **INSERT INTO t1 SELECT xxx** statement, a column in table **t1** needs to call the `nextval` function. If maximum number of connections on the GTM is 8192, this type of pushed statements occupies too many GTM connections. Therefore, the number of concurrent connections for these statements is limited to 7000 divided by the number of cluster DN. The other connections are reserved for other statements.

Return type: numeric

The `nextval` function can be called in either of the following ways: (In example 2, the ORA syntax is supported. Currently, the sequence name cannot contain a dot.)

```
gaussdb=# CREATE SEQUENCE seqDemo;
-- Example 1:
gaussdb=# SELECT nextval('seqDemo');
nextval

 1
(1 row)
-- Example 2:
gaussdb=# SELECT seqDemo.nextval;
nextval

 2
(1 row)
gaussdb=# DROP SEQUENCE seqDemo;
```

- `currval(regclass)`

Description: Returns the last value returned by `nextval` in the current session. If `nextval` has not been called for the specified sequence in the current session, an error is reported when `currval` is called. By default, this function is disabled. To enable it, set **enable\_beta\_features** to **true**. After **enable\_beta\_features** is set to **true**, the `nextval()` function will not be pushed down.

Return type: numeric

The `currval` function can be called in either of the following ways: (In example 2, the ORA syntax is supported. Currently, the sequence name cannot contain a dot.)

```
gaussdb=# CREATE SEQUENCE seq1;
gaussdb=# SELECT nextval('seq1');
gaussdb=# SET enable_beta_features = true;
-- Example 1:
gaussdb=# SELECT currval('seq1');
currval

 1
(1 row)
-- Example 2:
gaussdb=# SELECT seq1.currval seq1;
seq1

 1
(1 row)
gaussdb=# DROP SEQUENCE seq1;
gaussdb=# SET enable_beta_features = false;
```

- `lastval()`

Description: Returns the last value returned by `nextval` in the current session. This function is equivalent to `currval`, except that it does not use sequence names as parameters. It fetches the sequence used by `nextval` last time in the current session. If `nextval` has not been called in the current session, an error is reported when `lastval` is called.

By default, this function is disabled. To enable it, set **enable\_beta\_features** or **lastval\_supported** to **true**. After this function is enabled, `nextval()` will not be pushed down.

Return type: numeric

Example:

```
gaussdb=# CREATE SEQUENCE seq1;
gaussdb=# SELECT nextval('seq1');
gaussdb=# SET enable_beta_features = true;
gaussdb=# SELECT lastval();
lastval

 1
(1 row)
gaussdb=# DROP SEQUENCE seq1;
gaussdb=# SET enable_beta_features = false;
```

- `setval(regclass, bigint)`

Sets the current value of a sequence.

Return type: numeric

Example:

```
gaussdb=# CREATE SEQUENCE seqDemo;
gaussdb=# SELECT nextval('seqDemo');
gaussdb=# SELECT setval('seqDemo',3);
setval
```

```

 3
(1 row)
gaussdb=# DROP SEQUENCE seqDemo;
```

- **setval(regclass, numeric, Boolean)**

Sets the current value of a sequence and the is\_called sign.

Return type: numeric

Example:

```
gaussdb=# CREATE SEQUENCE seqDemo;
gaussdb=# SELECT nextval('seqDemo');
gaussdb=# SELECT setval('seqDemo',5,true);
setval
```

```

 5
(1 row)
gaussdb=# DROP SEQUENCE seqDemo;
```

#### NOTE

The current session and GTM will take effect immediately after setval is performed. If other sessions have buffered sequence values, setval will take effect only after the values are used up. Therefore, to prevent sequence value conflicts, you are advised to perform setval with caution.

Because the sequence is non-transactional, changes made by setval will not be canceled when a transaction rolled back.

- **pg\_sequence\_last\_value(sequence\_oid oid, OUT cache\_value int16, OUT last\_value int16)**  
Description: Obtains the parameters of a specified sequence, including the cache value and current value.  
Return type: int16, int16
- **gs\_get\_sequence\_last\_value(sequence\_oid oid, OUT cache\_value int16, OUT last\_value int16)**  
Description: Obtains the parameters of a specified sequence, including the cache value and current value. This function returns **NULL** for an unauthorized sequence.  
Return type: int16, int16

## 7.5.16 Array Functions and Operators

### Array Operators

- **=**  
Description: Specifies whether two arrays are equal.  
Example:  
gaussdb=# SELECT ARRAY[1,1,2,1,3,1]::int[] = ARRAY[1,2,3] AS RESULT ;  
result  
-----  
t  
(1 row)
- **<>**  
Description: Specifies whether two arrays are not equal.  
Example:  
gaussdb=# SELECT ARRAY[1,2,3] <> ARRAY[1,2,4] AS RESULT;  
result



Description: Specifies whether an array overlaps another (have common elements).

Example:

```
gaussdb=# SELECT ARRAY[1,4,3] && ARRAY[2,1] AS RESULT;
result

t
(1 row)
```

- ||

Description: Array-to-array concatenation.

Example:

```
gaussdb=# SELECT ARRAY[1,2,3] || ARRAY[4,5,6] AS RESULT;
result

{1,2,3,4,5,6}
(1 row)
gaussdb=# SELECT ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]] AS RESULT;
result

{{1,2,3},{4,5,6},{7,8,9}}
(1 row)
```

- ||

Description: Element-to-array concatenation.

Example:

```
gaussdb=# SELECT 3 || ARRAY[4,5,6] AS RESULT;
result

{3,4,5,6}
(1 row)
```

- ||

Description: Array-to-element concatenation.

Example:

```
gaussdb=# SELECT ARRAY[4,5,6] || 7 AS RESULT;
result

{4,5,6,7}
(1 row)
```

Array comparisons compare the array contents element-by-element, using the default B-tree comparison function for the element data type. In multidimensional arrays, the elements are accessed in row-major order. If the contents of two arrays are equal but the dimensionality is different, the first difference in the dimensionality information determines the sort order.

## Array Functions

- `array_append(anyarray, anyelement)`

Description: Appends an element to the end of an array, and only supports dimension-1 arrays.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_append(ARRAY[1,2], 3) AS RESULT;
result

{1,2,3}
(1 row)
```

- `array_prepend(anyelement, anyarray)`

Description: Appends an element to the beginning of an array, and only supports dimension-1 arrays.

Return type: `anyarray`

Example:

```
gaussdb=# SELECT array_prepend(1, ARRAY[2,3]) AS RESULT;
result

{1,2,3}
(1 row)
```

- `array_cat(anyarray, anyarray)`

Description: Concatenates two arrays, and supports multi-dimensional arrays.

Return type: `anyarray`

Example:

```
gaussdb=# SELECT array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;
result

{1,2,3,4,5}
(1 row)

gaussdb=# SELECT array_cat(ARRAY[[1,2],[4,5]], ARRAY[6,7]) AS RESULT;
result

{{1,2},{4,5},{6,7}}
(1 row)
```

- `array_union(anyarray, anyarray)`

Description: Concatenates two arrays. Only one-dimensional arrays are supported. If an input parameter is **NULL**, another input parameter is returned.

Return type: `anyarray`

Example:

```
gaussdb=# SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
result

{1,2,3,3,4,5}
(1 row)

gaussdb=# SELECT array_union(ARRAY[1,2,3], NULL) AS RESULT;
result

{1,2,3}
(1 row)
```

- `array_union_distinct(anyarray, anyarray)`

Description: Concatenates two arrays and deduplicates them. Only one-dimensional arrays are supported. If an input parameter is **NULL**, another input parameter is returned.

Return type: `anyarray`

Example:

```
gaussdb=# SELECT array_union_distinct(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
result

{1,2,3,4,5}
(1 row)

gaussdb=# SELECT array_union_distinct(ARRAY[1,2,3], NULL) AS RESULT;
result
```

```

{1,2,3}
(1 row)
```

- `array_intersect(anyarray, anyarray)`

Description: Intersects two arrays. Only one-dimensional arrays are supported. If any input parameter is **NULL**, **NULL** is returned.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
result
```

```

{3}
(1 row)
```

```
gaussdb=# SELECT array_intersect(ARRAY[1,2,3], NULL) AS RESULT;
result
```

```

(1 row)
```

- `array_intersect_distinct(anyarray, anyarray)`

Description: Intersects two arrays and deduplicates them. Only one-dimensional arrays are supported. If any input parameter is **NULL**, **NULL** is returned.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_intersect_distinct(ARRAY[1,2,2], ARRAY[2,2,4,5]) AS RESULT;
result
```

```

{2}
(1 row)
```

```
gaussdb=# SELECT array_intersect_distinct(ARRAY[1,2,3], NULL) AS RESULT;
result
```

```

(1 row)
```

- `array_except(anyarray, anyarray)`

Description: Calculates the difference between two arrays. Only one-dimensional arrays are supported. If the first input parameter is **NULL**, **NULL** is returned. If the second input parameter is **NULL**, the first input parameter is returned.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
result
```

```

{1,2}
(1 row)
```

```
gaussdb=# SELECT array_except(ARRAY[1,2,3], NULL) AS RESULT;
result
```

```

{1,2,3}
(1 row)
```

```
gaussdb=# SELECT array_except(NULL, ARRAY[3,4,5]) AS RESULT;
result
```

```

```

- (1 row)
- array\_except\_distinct(anyarray, anyarray)**

Description: Calculates the difference between two arrays and deduplicates them. Only one-dimensional arrays are supported. If the first input parameter is **NULL**, **NULL** is returned. If the second input parameter is **NULL**, the first input parameter is returned.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_except_distinct(ARRAY[1,2,2,3], ARRAY[3,4,5]) AS RESULT;
result

{1,2}
(1 row)

gaussdb=# SELECT array_except_distinct(ARRAY[1,2,3], NULL) AS RESULT;
result

{1,2,3}
(1 row)

gaussdb=# SELECT array_except_distinct(NULL, ARRAY[3,4,5]) AS RESULT;
result

(1 row)
```
  - array\_ndims(anyarray)**

Description: Returns the number of dimensions of an array.

Return type: int

Example:

```
gaussdb=# SELECT array_ndims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
result

2
(1 row)
```
  - array\_dims(anyarray)**

Description: Returns a text representation of array's dimensions.

Return type: text

Example:

```
gaussdb=# SELECT array_dims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
result

[1:2][1:3]
(1 row)
```
  - array\_length(anyarray, int)**

Description: Returns the length of the requested array dimension.

Return type: int

Example:

```
gaussdb=# SELECT array_length(array[1,2,3], 1) AS RESULT;
result

3
(1 row)
```
  - array\_lower(anyarray, int)**

Description: Returns lower bound of the requested array dimension.

Return type: int

Example:

```
gaussdb=# SELECT array_lower('[0:2]={1,2,3}::int[], 1) AS RESULT;
result

 0
(1 row)
```

- `array_sort(anyarray)`

Description: Returns an array in ascending order.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_sort(ARRAY[5,1,3,6,2,7]) AS RESULT;
result

{1,2,3,5,6,7}
(1 row)
```

- `array_upper(anyarray, int)`

Description: Returns upper bound of the requested array dimension.

Return type: int

Example:

```
gaussdb=# SELECT array_upper(ARRAY[1,8,3,7], 1) AS RESULT;
result

 4
(1 row)
```

- `array_to_string(anyarray, text [, text])`

Description: Uses the first **text** as the new delimiter and the second **text** to replace **NULL** values.

Return type: text

Example:

```
gaussdb=# SELECT array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '**') AS RESULT;
result

1,2,3,**5
(1 row)
```

- `array_delete(anyarray)`

Description: Clears elements in an array and returns an empty array of the same type.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_delete(ARRAY[1,8,3,7]) AS RESULT;
result

{}
(1 row)
```

- `array_deleteidx(anyarray, int)`

Description: Deletes specified index elements from an array and returns an array consisting of the remaining elements.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_deleteidx(ARRAY[1,2,3,4,5], 1) AS RESULT;
result
```

```

{2,3,4,5}
(1 row)
```

 **NOTE**

array\_deleteidx(anyarray, int): This function is disabled when the value of **a\_format\_version** is **10c** and the value of **a\_format\_dev\_version** is **s1** in the ORA-compatible database.

- array\_extendnull(anyarray, int)

Description: Adds a specified number of null elements to the end of an array.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_extendnull(ARRAY[1,8,3,7],1) AS RESULT;
result

{1,8,3,7,null}
(1 row)
```

- array\_extendnull(anyarray, int, int)

Description: Adds a specified number of elements with a specified index to the end of an array.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_extendnull(ARRAY[1,8,3,7],2,2) AS RESULT;
result

{1,8,3,7,8,8}
(1 row)
```

 **NOTE**

array\_extendnull(anyarray, int, int): This function takes effect when the value of **a\_format\_version** is **10c** and the value of **a\_format\_dev\_version** is **s1** in an ORA-compatible database.

- array\_trim(anyarray, int)

Description: Deletes a specified number of elements from the end of an array.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_trim(ARRAY[1,8,3,7],1) AS RESULT;
result

{1,8,3}
(1 row)
```

- array\_exists(anyarray, int)

Description: Checks whether the second parameter is a valid index of an array.

Return type: Boolean

Example:

```
gaussdb=# SELECT array_exists(ARRAY[1,8,3,7],1) AS RESULT;
result

t
(1 row)
```

- array\_next(anyarray, int)

Description: Returns the index of the element following a specified index in an array based on the second input parameter.

Return type: int

Example:

```
gaussdb=# SELECT array_next(ARRAY[1,8,3,7],1) AS RESULT;
result

2
(1 row)
```

- `array_prior(anyarray, int)`

Description: Returns the index of the element followed by a specified index in an array based on the second input parameter.

Return type: int

Example:

```
gaussdb=# SELECT array_prior(ARRAY[1,8,3,7],2) AS RESULT;
result

1
(1 row)
```

- `string_to_array(text, text [, text])`

Description: Uses the second **text** as the new delimiter and the third **text** as the substring to be replaced by **NULL** values. A substring can be replaced by **NULL** values only when it is the same as the third **text**.

Return type: text[]

Example:

```
gaussdb=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
result

{xx,NULL,zz}
(1 row)
gaussdb=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
result

{xx,y,zz}
(1 row)
```

- `unnest(anyarray)`

Description: Expands an array to a set of rows.

Return type: setof anyelement

Example:

```
gaussdb=# SELECT unnest(ARRAY[1,2]) AS RESULT;
result

1
2
(2 rows)
```

In **string\_to\_array**, if the delimiter parameter is **NULL**, each character in the input string will become a separate element in the resulting array. If the delimiter is an empty string, then the entire input string is returned as a one-element array. Otherwise the input string is split at each occurrence of the delimiter string.

In **string\_to\_array**, if the null-string parameter is omitted or **NULL**, none of the substrings of the input will be replaced by **NULL**.

In **array\_to\_string**, if the null-string parameter is omitted or **NULL**, any null elements in the array are simply skipped and not represented in the output string.

- `_pg_keysequal`  
Description: Checks whether two smallint arrays are the same.  
Parameter: `smallint[], smallint[]`  
Return type: Boolean

 **NOTE**

This function exists in the `information_schema` namespace.

## 7.5.17 Range Functions and Operators

### Range Operators

- `=`  
Description: Equals  
Example:

```
gaussdb=# SELECT int4range(1,5) = '[1,4]::int4range AS RESULT;
result

t
(1 row)
```
- `<>`  
Description: Does not equal  
Example:

```
gaussdb=# SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;
result

t
(1 row)
```
- `<`  
Description: Is less than  
Example:

```
gaussdb=# SELECT int4range(1,10) < int4range(2,3) AS RESULT;
result

t
(1 row)
```
- `>`  
Description: Is greater than  
Example:

```
gaussdb=# SELECT int4range(1,10) > int4range(1,5) AS RESULT;
result

t
(1 row)
```
- `<=`  
Description: Is less than or equals  
Example:

```
gaussdb=# SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
result
```

- t  
(1 row)

  - >=

Description: Is greater than or equals

Example:

```
gaussdb=# SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
result

t
(1 row)
```
  - @>

Description: Contains range

Example:

```
gaussdb=# SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
result

t
(1 row)
```
  - @>

Description: Contains element

Example:

```
gaussdb=# SELECT '[2011-01-01,2011-03-01]::tsrange @> '2011-01-10'::timestamp AS RESULT;
result

t
(1 row)
```
  - <@

Description: Range is contained by

Example:

```
gaussdb=# SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
result

t
(1 row)
```
  - <@

Description: Element is contained by

Example:

```
gaussdb=# SELECT 42 <@ int4range(1,7) AS RESULT;
result

f
(1 row)
```
  - &&

Description: Overlaps (has points in common)

Example:

```
gaussdb=# SELECT int8range(3,7) && int8range(4,12) AS RESULT;
result

t
(1 row)
```
  - <<

Description: Strictly left of

Example:

```
gaussdb=# SELECT int8range(1,10) << int8range(100,110) AS RESULT;
result

t
(1 row)
```

- >>

Description: Strictly right of

Example:

```
gaussdb=# SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
result

t
(1 row)
```

- &<

Description: Does not extend to the right of

Example:

```
gaussdb=# SELECT int8range(1,20) &< int8range(18,20) AS RESULT;
result

t
(1 row)
```

- &>

Description: Does not extend to the left of

Example:

```
gaussdb=# SELECT int8range(7,20) &> int8range(5,10) AS RESULT;
result

t
(1 row)
```

- -|-

Description: Is adjacent to

Example:

```
gaussdb=# SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;
result

t
(1 row)
```

- +

Description: Union

Example:

```
gaussdb=# SELECT numrange(5,15) + numrange(10,20) AS RESULT;
result

[5,20)
(1 row)
```

- \*

Description: Intersection

Example:

```
gaussdb=# SELECT int8range(5,15) * int8range(10,20) AS RESULT;
result

[10,15)
(1 row)
```

- -  
Description: Difference

Example:

```
gaussdb=# SELECT int8range(5,15) - int8range(10,20) AS RESULT;
result

[5,10)
(1 row)
```

The simple comparison operators <, >, <=, and >= compare the lower bounds first, and only if those are equal, compare the upper bounds.

The <<, >>, and -|- operators always return false when an empty range is involved; that is, an empty range is not considered to be either before or after any other range.

The union and difference operators will fail if the resulting range would need to contain two disjoint sub-ranges.

## Range Functions

The lower and upper functions return null if the range is empty or the requested bound is infinite. The lower\_inc, upper\_inc, lower\_inf, and upper\_inf functions all return false for an empty range.

- numrange(numeric, numeric, [text])

Description: Specifies a range.

Return type: Range's element type

Example:

```
gaussdb=# SELECT numrange(1.1,2.2) AS RESULT;
result

[1.1,2.2)
(1 row)
gaussdb=# SELECT numrange(1.1,2.2, '(') AS RESULT;
result

(1.1,2.2)
(1 row)
```

- lower(anyrange)

Description: Lower bound of range

Return type: Range's element type

Example:

```
gaussdb=# SELECT lower(numrange(1.1,2.2)) AS RESULT;
result

1.1
(1 row)
```

- upper(anyrange)

Description: Upper bound of range

Return type: range's element type

Example:

```
gaussdb=# SELECT upper(numrange(1.1,2.2)) AS RESULT;
result

```

2.2  
(1 row)

- **isempty(anyrange)**

Description: Is the range empty?

Return type: Boolean

Example:

```
gaussdb=# SELECT isempty(numrange(1.1,2.2)) AS RESULT;
result

f
(1 row)
```

- **lower\_inc(anyrange)**

Description: Is the lower bound inclusive?

Return type: Boolean

Example:

```
gaussdb=# SELECT lower_inc(numrange(1.1,2.2)) AS RESULT;
result

t
(1 row)
```

- **upper\_inc(anyrange)**

Description: Is the upper bound inclusive?

Return type: Boolean

Example:

```
gaussdb=# SELECT upper_inc(numrange(1.1,2.2)) AS RESULT;
result

f
(1 row)
```

- **lower\_inf(anyrange)**

Description: Is the lower bound infinite?

Return type: Boolean

Example:

```
gaussdb=# SELECT lower_inf('()::daterange) AS RESULT;
result

t
(1 row)
```

- **upper\_inf(anyrange)**

Description: Is the upper bound infinite?

Return type: Boolean

Example:

```
gaussdb=# SELECT upper_inf('()::daterange) AS RESULT;
result

t
(1 row)
```

- **elem\_contained\_by\_range(anelement, anyrange)**

Description: Determines whether an element is within the range.

Return type: Boolean

Example:

```
gaussdb=# SELECT elem_contained_by_range('2', numrange(1.1,2.2));
elem_contained_by_range

t
(1 row)
```

## 7.5.18 Aggregate Functions

### Aggregate Functions

- `sum(expression)`

Description: Specifies the sum of expressions across all input values.

Return type:

Generally, same as the argument data type. In the following cases, type conversion occurs:

- BIGINT for SMALLINT or INT arguments
- NUMBER for BIGINT arguments
- DOUBLE PRECISION for floating-point arguments

Example:

```
gaussdb=# CREATE TABLE tab(a int);
CREATE TABLE
gaussdb=# INSERT INTO tab values(1);
INSERT 0 1
gaussdb=# INSERT INTO tab values(2);
INSERT 0 1
gaussdb=# SELECT sum(a) FROM tab;
sum

3
(1 row)
```

- `max(expression)`

Description: Specifies the maximum value of expression across all input values.

Parameter type: any array, numeric, string, or date/time type

Return type: same as the argument type

Example:

```
gaussdb=# CREATE TABLE max_t1(a int, b int);
gaussdb=# INSERT INTO max_t1 VALUES(1,2),(2,3),(3,4),(4,5);
gaussdb=# SELECT MAX(a) FROM max_t1;
max

4
(1 row)
gaussdb=# DROP TABLE max_t1;
```

- `min(expression)`

Description: Specifies the minimum value of expression across all input values.

Parameter type: any array, numeric, string, or date/time type

Return type: same as the argument type

Example:

```
gaussdb=# CREATE TABLE min_t1(a int, b int);
gaussdb=# INSERT INTO min_t1 VALUES(1,2),(2,3),(3,4),(4,5);
gaussdb=# SELECT MIN(a) FROM min_t1;
 min

 1
(1 row)
gaussdb=# DROP TABLE min_t1;
```

- **avg(expression)**

Description: Specifies the average (arithmetic mean) of all input values.

Return type:

NUMBER for any integer-type argument.

DOUBLE PRECISION for floating-point parameters.

otherwise the same as the argument data type.

Example:

```
gaussdb=# CREATE TABLE avg_t1(a int, b int);
gaussdb=# INSERT INTO avg_t1 VALUES(1,2),(2,3),(3,4),(4,5);
gaussdb=# SELECT AVG(a) FROM avg_t1;
 avg

2.5000000000000000
(1 row)
gaussdb=# DROP TABLE avg_t1;
```

- **count(expression)**

Description: Specifies the number of input rows for which the value of the expression is not null.

Return type: bigint

Operations on XML data are supported.

Example:

```
gaussdb=# CREATE TABLE count_t1(a int, b int);
gaussdb=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
gaussdb=# SELECT COUNT(a) FROM count_t1;
 count

 4
(1 row)
gaussdb=# DROP TABLE count_t1;
```

- **count(\*)**

Description: Returns the number of input rows.

Return type: bigint

Operations on XML data are supported.

Example:

```
gaussdb=# CREATE TABLE count_t1(a int, b int);
gaussdb=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
gaussdb=# SELECT COUNT(*) FROM count_t1;
```

```
count

 5
(1 row)
```

```
gaussdb=# DROP TABLE count_t1;
```

- **array\_agg(expression)**

Description: Concatenates input values, including nulls, into an array.

Return type: array of the argument type

Operations on XML data are supported.

Example:

```
gaussdb=# CREATE TABLE array_agg_t1(a int, b int);
```

```
gaussdb=# INSERT INTO array_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
```

```
gaussdb=# SELECT ARRAY_AGG(a) FROM array_agg_t1;
array_agg
```

```

{NULL,1,2,3,4}
(1 row)
```

```
gaussdb=# DROP TABLE array_agg_t1;
```

- **string\_agg(expression, delimiter)**

Description: Concatenates input values into a string, separated by delimiter.

Return type: same as the argument type

Operations on XML data that is explicitly converted to the character type are supported.

Example:

```
gaussdb=# CREATE TABLE string_agg_t1(a int, b int);
```

```
gaussdb=# INSERT INTO string_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
```

```
gaussdb=# SELECT STRING_AGG(a,',') FROM string_agg_t1;
string_agg
```

```

1;2;3;4
(1 row)
```

```
gaussdb=# DROP TABLE string_agg_t1;
```

- **listagg(expression [, delimiter]) WITHIN GROUP(ORDER BY order-list)**

Description: Sorts aggregation column data according to the mode specified by **WITHIN GROUP** and concatenates the data to a string using the specified delimiter.

- **expression:** Required. It specifies an aggregation column name or a column-based valid expression. It does not support the **DISTINCT** keyword and the **VARIADIC** parameter.
- **delimiter:** Optional. It specifies a delimiter, which can be a string constant or a deterministic expression based on a group of columns. The default value is empty.
- **order-list:** Required. It specifies the sorting mode in a group.

Return type: text

 **NOTE**

listagg is a column-to-row aggregate function, compatible with Oracle Database 11g Release 2. You can specify the OVER clause as a window function. When **listagg** is used as a window function, the **OVER** clause does not support the window sorting or framework of **ORDER BY**, to avoid ambiguity in **listagg** and **ORDER BY** of the **WITHIN GROUP** clause.

Example:

The aggregation column is of the text character set type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b text);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'a1'),(1,'b2'),(1,'c3'),(2,'d4'),(2,'e5'),(3,'f6');

gaussdb=# SELECT a,LISTAGG(b,') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
a | listagg
---+-----
1 | b2;c3
2 | d4;e5
3 | f6
 | a1
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

The aggregation column is of the integer type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b int);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT a,LISTAGG(b,') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
a | listagg
---+-----
1 | 2;3
2 | 4;5
3 | 6
 | 1
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

The aggregation column is of the floating point type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b float);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,1.111),(1,2.222),(1,3.333),(2,4.444),(2,5.555),
(3,6.666);

gaussdb=# SELECT a,LISTAGG(b,') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
a | listagg
---+-----
1 | 2.222000;3.333000
2 | 4.444000;5.555000
3 | 6.666000
 | 1.111000
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

The aggregation column is of the time type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b timestamp);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'2000-01-01'),(1,'2000-02-02'),(1,'2000-03-03'),
(2,'2000-04-04'),(2,'2000-05-05'),(3,'2000-06-06');

gaussdb=# SELECT a,LISTAGG(b,') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
a | listagg
---+-----
```

```
1 | 2000-02-02 00:00:00;2000-03-03 00:00:00
2 | 2000-04-04 00:00:00;2000-05-05 00:00:00
3 | 2000-06-06 00:00:00
 | 2000-01-01 00:00:00
(4 rows)
```

```
gaussdb=# DROP TABLE listagg_t1;
```

The aggregation column is of the time interval type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);
```

```
gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');
```

```
gaussdb=# SELECT a,LISTAGG(b,;) WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
```

```
a | listagg
---+-----
1 | 2 days;3 days
2 | 4 days;5 days
3 | 6 days
 | 1 day
(4 rows)
```

```
gaussdb=# DROP TABLE listagg_t1;
```

By default, the delimiter is empty.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);
```

```
gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');
```

```
gaussdb=# SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
```

```
a | listagg
---+-----
1 | 2 days3 days
2 | 4 days5 days
3 | 6 days
 | 1 day
(4 rows)
```

```
gaussdb=# DROP TABLE listagg_t1;
```

When listagg is used as a window function, the OVER clause does not support the window sorting of ORDER BY, and the listagg column is an ordered aggregation of the corresponding groups.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);
```

```
gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');
```

```
gaussdb=# SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) OVER(PARTITION BY a) FROM
listagg_t1;
```

```
a | listagg
---+-----
1 | 2 days3 days
1 | 2 days3 days
2 | 4 days5 days
2 | 4 days5 days
3 | 6 days
 | 1 day
(6 rows)
```

```
gaussdb=# DROP TABLE listagg_t1;
```

- **wm\_concat(expression)**

Description: Concatenates column data into a string separated by commas (,).

Return type: same as the argument type

 NOTE

**wm\_concat** is an ORA compatibility requirement. Currently, this function has been canceled in the latest ORA version and replaced by the listagg function. You can also use the string\_agg function. For details, see the description of the two functions.

- covar\_pop(Y, X)

Description: Specifies the overall covariance.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE covar_pop_t1(a int, b int);
gaussdb=# INSERT INTO covar_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
gaussdb=# SELECT COVAR_POP(a,b) FROM covar_pop_t1;
 covar_pop

 100
(1 row)
gaussdb=# DROP TABLE covar_pop_t1;
```

- covar\_samp(Y, X)

Description: Specifies the sample covariance.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE covar_samp_t1(a int, b int);
gaussdb=# INSERT INTO covar_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
gaussdb=# SELECT COVAR_SAMP(a,b) FROM covar_samp_t1;
 covar_samp

 125
(1 row)
gaussdb=# DROP TABLE covar_samp_t1;
```

- stddev\_pop(expression)

Description: Specifies the overall standard deviation.

Return type: **double precision** for floating-point arguments, otherwise **numeric**

Example:

```
gaussdb=# CREATE TABLE stddev_pop_t1(a int, b int);
gaussdb=# INSERT INTO stddev_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
gaussdb=# SELECT STDDEV_POP(a) FROM stddev_pop_t1;
 stddev_pop

7.4833147735478828
(1 row)
gaussdb=# DROP TABLE stddev_pop_t1;
```

- stddev\_samp(expression)

Description: Specifies the sample standard deviation of the input values.

Return type: **double precision** for floating-point arguments, otherwise **numeric**

Example:

```
gaussdb=# CREATE TABLE stddev_samp_t1(a int, b int);
gaussdb=# INSERT INTO stddev_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
gaussdb=# SELECT STDDEV_SAMP(a) FROM stddev_samp_t1;
 stddev_samp

8.3666002653407555
(1 row)
gaussdb=# DROP TABLE stddev_samp_t1;
```

- **var\_pop(expression)**

Description: Specifies the population variance of the input values (square of the population standard deviation).

Return type: DOUBLE PRECISION for floating-point arguments, otherwise NUMERIC

Example:

```
gaussdb=# CREATE TABLE var_pop_t1(a int, b int);
gaussdb=# INSERT INTO var_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
gaussdb=# SELECT VAR_POP(a) FROM var_pop_t1;
 var_pop

56.0000000000000000
(1 row)
gaussdb=# DROP TABLE var_pop_t1;
```

- **var\_samp(expression)**

Description: Specifies the sample variance of the input values (square of the sample standard deviation).

Return type: DOUBLE PRECISION for floating-point arguments, otherwise NUMERIC

Example:

```
gaussdb=# CREATE TABLE var_samp_t1(a int, b int);
gaussdb=# INSERT INTO var_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
gaussdb=# SELECT VAR_SAMP(a) FROM var_samp_t1;
 var_samp

70.0000000000000000
(1 row)
gaussdb=# DROP TABLE var_samp_t1;
```

- **bit\_and(expression)**

Description: bitwise AND of all non-null input values, or null if none

Return type: same as the argument type

Example:

```
gaussdb=# CREATE TABLE bit_and_t1(a int, b int);
gaussdb=# INSERT INTO bit_and_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT BIT_AND(a) FROM bit_and_t1;
 bit_and

0
(1 row)
```

```
gaussdb=# DROP TABLE bit_and_t1;
```

- **bit\_or(expression)**

Description: bitwise OR of all non-null input values, or null if none

Return type: same as the argument type

Example:

```
gaussdb=# CREATE TABLE bit_or_t1(a int, b int);
```

```
gaussdb=# INSERT INTO bit_or_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
gaussdb=# SELECT BIT_OR(a) FROM bit_or_t1;
```

```
bit_or

 3
(1 row)
```

```
gaussdb=# DROP TABLE bit_or_t1;
```

- **bool\_and(expression)**

Description: Its value is **true** if all input values are **true**, otherwise **false**.

Return type: Boolean

Example:

```
gaussdb=# SELECT bool_and(100 <2500);
```

```
bool_and

 t
(1 row)
```

- **bool\_or(expression)**

Description: Its value is **true** if at least one input value is **true**, otherwise **false**.

Return type: Boolean

Example:

```
gaussdb=# SELECT bool_or(100 <2500);
```

```
bool_or

 t
(1 row)
```

- **corr(Y, X)**

Description: Specifies the correlation coefficient.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE corr_t1(a int, b int);
```

```
gaussdb=# INSERT INTO corr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
gaussdb=# SELECT CORR(a,b) FROM corr_t1;
```

```
corr

.944911182523068
(1 row)
```

```
gaussdb=# DROP TABLE corr_t1;
```

- **every(expression)**

Description: Equivalent to **bool\_and**

Return type: Boolean

Example:

```
gaussdb=# SELECT every(100 <2500);
every

t
(1 row)
```

- **rank(expression)**

Description: The tuples in different groups are sorted non-consecutively by **expression**.

Return type: bigint

Example:

```
gaussdb=# CREATE TABLE rank_t1(a int, b int);
gaussdb=# INSERT INTO rank_t1 VALUES(NULL,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);
gaussdb=# SELECT a,b,RANK() OVER(PARTITION BY a ORDER BY b) FROM rank_t1;
a | b | rank
---+---+-----
1 | 2 | 1
1 | 3 | 2
2 | 4 | 1
2 | 5 | 2
3 | 6 | 1
 | 1 | 1
(6 rows)
gaussdb=# DROP TABLE rank_t1;
```

- **regr\_avgx(Y, X)**

Description: Specifies the average of the independent variable (**sum(X)/N**).

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE regr_t1(a int, b int);
gaussdb=# INSERT INTO regr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT REGR_AVGX(a,b) FROM regr_t1;
regr_avgx

4
(1 row)
gaussdb=# DROP TABLE regr_t1;
```

- **regr\_avgy(Y, X)**

Description: Specifies the average of the dependent variable (**sum(Y)/N**).

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE regr_avgy_t1(a int, b int);
gaussdb=# INSERT INTO regr_avgy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT REGR_AVGY(a,b) FROM regr_avgy_t1;
regr_avgy

1.8
(1 row)
gaussdb=# DROP TABLE regr_avgy_t1;
```

- **regr\_count(Y, X)**

Description: Specifies the number of input rows in which both expressions are non-null.

Return type: bigint

Example:

```
gaussdb=# CREATE TABLE regr_count_t1(a int, b int);
gaussdb=# INSERT INTO regr_count_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT REGR_COUNT(a,b) FROM regr_count_t1;
regr_count

 5
(1 row)
gaussdb=# DROP TABLE regr_count_t1;
```

- **regr\_intercept(Y, X)**

Description: Specifies the y-intercept of the least-squares-fit linear equation determined by the (X, Y) pairs.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE regr_intercept_t1(a int, b int);
gaussdb=# INSERT INTO regr_intercept_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT REGR_INTERCEPT(b,a) FROM regr_intercept_t1;
regr_intercept

.785714285714286
(1 row)
gaussdb=# DROP TABLE regr_intercept_t1;
```

- **regr\_r2(Y, X)**

Description: Specifies the square of the correlation coefficient.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE regr_r2_t1(a int, b int);
gaussdb=# INSERT INTO regr_r2_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT REGR_R2(b,a) FROM regr_r2_t1;
regr_r2

.892857142857143
(1 row)
gaussdb=# DROP TABLE regr_r2_t1;
```

- **regr\_slope(Y, X)**

Description: Specifies the slope of the least-squares-fit linear equation determined by the (X, Y) pairs.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE regr_slope_t1(a int, b int);
gaussdb=# INSERT INTO regr_slope_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT REGR_SLOPE(b,a) FROM regr_slope_t1;
regr_slope
```

```

1.78571428571429
(1 row)
```

```
gaussdb=# DROP TABLE regr_slope_t1;
```

- **regr\_sxx(Y, X)**

Description:  **$\sum(X^2) - \sum(X)^2/N$**  (sum of squares of the independent variables)

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE regr_sxx_t1(a int, b int);
```

```
gaussdb=# INSERT INTO regr_sxx_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
gaussdb=# SELECT REGR_SXX(b,a) FROM regr_sxx_t1;
regr_sxx
```

```

2.8
(1 row)
```

```
gaussdb=# DROP TABLE regr_sxx_t1;
```

- **regr\_sxy(Y, X)**

Description:  **$\sum(X*Y) - \sum(X) * \sum(Y)/N$**  (sum of products of independent times dependent variable)

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE regr_sxy_t1(a int, b int);
```

```
gaussdb=# INSERT INTO regr_sxy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
gaussdb=# SELECT REGR_SXY(b,a) FROM regr_sxy_t1;
regr_sxy
```

```

5
(1 row)
```

```
gaussdb=# DROP TABLE regr_sxy_t1;
```

- **regr\_syy(Y, X)**

Description:  **$\sum(Y^2) - \sum(Y)^2/N$**  (sum of squares of the dependent variable)

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE regr_syy_t1(a int, b int);
```

```
gaussdb=# INSERT INTO regr_syy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
gaussdb=# SELECT REGR_SYY(b,a) FROM regr_syy_t1;
regr_syy
```

```

10
(1 row)
```

```
gaussdb=# DROP TABLE regr_syy_t1;
```

- **stddev(expression)**

Description: Specifies the alias of **stddev\_samp**.

Return type: **double precision** for floating-point arguments, otherwise **numeric**

**Example:**

```
gaussdb=# CREATE TABLE stddev_t1(a int, b int);
gaussdb=# INSERT INTO stddev_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT STDDEV(a) FROM stddev_t1;
 stddev

.83666002653407554798
(1 row)
gaussdb=# DROP TABLE stddev_t1;
```

- **variance(expression,ression)**

Description: Specifies the alias of **var\_samp**.

Return type: **double precision** for floating-point arguments, otherwise **numeric**

**Example:**

```
gaussdb=# CREATE TABLE variance_t1(a int, b int);
gaussdb=# INSERT INTO variance_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT VARIANCE(a) FROM variance_t1;
 variance

.70000000000000000000000000000000
(1 row)
gaussdb=# DROP TABLE variance_t1;
```

- **spread**

Description: Calculates the difference between the maximum value and minimum value in a certain period.

Parameter: real

Return type: real

- **pivot\_func(anelement)**

Description: Returns the only non-null value in a column. If there are two or more non-null values, an error is reported. **value** is an input parameter and can be of any type.

Return type: same as the input parameter type

 **NOTE**

This aggregate function is mainly used inside the pivot syntax.

```
gaussdb=# CREATE TABLE pivot_func_t1(a int, b int);
gaussdb=# INSERT INTO pivot_func_t1 VALUES (NULL,11),(1,2);
gaussdb=# SELECT PIVOT_FUNC(a) FROM pivot_func_t1;
 pivot_func

1
(1 row)
gaussdb=# DROP TABLE pivot_func_t1;
```

- **checksum(expression)**

Description: Returns the **CHECKSUM** value of all input values. This function can be used to check whether the data in the tables is the same before and after the backup, restoration, or migration of the GaussDB database

(databases other than GaussDB are not supported). Before and after database backup, database restoration, or data migration, you need to manually run SQL commands to obtain the execution results. Compare the obtained execution results to check whether the data in the tables before and after the backup or migration is the same.

**NOTE**

- For large tables, the execution of CHECKSUM function may take a long time.
  - If the CHECKSUM values of two tables are different, it indicates that the contents of the two tables are different. Using the hash function in the CHECKSUM function may incur conflicts. There is low possibility that two tables with different contents may have the same CHECKSUM value. The same problem may occur when CHECKSUM is used for columns.
  - If the time type is timestamp, timestamptz, or smalldatetime, ensure that the time zone settings are the same when calculating the CHECKSUM value.
- If the CHECKSUM value of a column is calculated and the column type can be changed to TEXT by default, set *expression* to the column name.
  - If the CHECKSUM value of a column is calculated and the column type cannot be converted to TEXT by default, set *expression* to *Column name::TEXT*.
  - If the CHECKSUM value of all columns is calculated, set *expression* to *Table name::TEXT*.

The following types of data can be converted into the TEXT type by default: char, name, int8, int2, int1, int4, raw, pg\_node\_tree, float4, float8, bpchar, varchar, nvarchar2, date, timestamp, timestamptz, numeric, and smalldatetime. Other types (for example, XML) need to be forcibly converted to TEXT.

Return type: numeric

Example:

The following shows the CHECKSUM value of a column that can be converted to the TEXT type by default:

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);
gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT CHECKSUM(a) FROM checksum_t1;
checksum

18126842830
(1 row)

gaussdb=# DROP TABLE checksum_t1;
```

The following shows the CHECKSUM value of a column that cannot be converted to the TEXT type by default. Note that the CHECKSUM parameter is set to *Column name::TEXT*.

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);
gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT CHECKSUM(a::TEXT) FROM checksum_t1;
checksum

18126842830
(1 row)

gaussdb=# DROP TABLE checksum_t1;
```

The following shows the CHECKSUM value of all columns in a table. Note that the CHECKSUM parameter is set to *Table name::TEXT*. The table name is not modified by its schema.

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);
gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
gaussdb=# SELECT CHECKSUM(checksum_t1::TEXT) FROM checksum_t1;
checksum

1116052226
(1 row)
gaussdb=# DROP TABLE checksum_t1;
```

## 7.5.19 Window Functions

### Window Functions

This statement is used together with the window function. The OVER clause is used for grouping data and sorting the elements in a group. Window functions are used for generating sequence numbers for the values in the group.

#### NOTE

ORDER BY in a window function must be followed by a column name. If it is followed by a number, the number is processed as a constant value and the target column is not ranked.

- RANK()

Description: The RANK function is used for generating non-consecutive sequence numbers for the values in each group. The same values have the same sequence number.

Return type: bigint

Example:

```
gaussdb=# CREATE TABLE rank_t1(a int, b int);
gaussdb=# INSERT INTO rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);
gaussdb=# SELECT a,b,RANK() OVER(PARTITION BY a ORDER BY b) FROM rank_t1;
a | b | rank
-----+-----
1 | 1 | 1
1 | 1 | 1
1 | 2 | 3
1 | 3 | 4
2 | 4 | 1
2 | 5 | 2
3 | 6 | 1
(7 rows)
gaussdb=# DROP TABLE rank_t1;
```

- ROW\_NUMBER()

Description: The ROW\_NUMBER function is used for generating consecutive sequence numbers for the values in each group. The same values have different sequence numbers.

Return type: bigint

Example:

```
gaussdb=# CREATE TABLE row_number_t1(a int, b int);
```

```
gaussdb=# INSERT INTO row_number_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,ROW_NUMBER() OVER(PARTITION BY a ORDER BY b) FROM row_number_t1;
a | b | row_number
-----+-----
1 | 1 | 1
1 | 1 | 2
1 | 2 | 3
1 | 3 | 4
2 | 4 | 1
2 | 5 | 2
3 | 6 | 1
(7 rows)

gaussdb=# DROP TABLE row_number_t1;
```

- **DENSE\_RANK()**

Description: The DENSE\_RANK function is used for generating consecutive sequence numbers for the values in each group. The same values have the same sequence number.

Return type: bigint

Example:

```
gaussdb=# CREATE TABLE dense_rank_t1(a int, b int);

gaussdb=# INSERT INTO dense_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,DENSE_RANK() OVER(PARTITION BY a ORDER BY b) FROM dense_rank_t1;
a | b | dense_rank
-----+-----
1 | 1 | 1
1 | 1 | 1
1 | 2 | 2
1 | 3 | 3
2 | 4 | 1
2 | 5 | 2
3 | 6 | 1
(7 rows)

gaussdb=# DROP TABLE dense_rank_t1;
```

- **PERCENT\_RANK()**

Description: The PERCENT\_RANK function is used for generating corresponding sequence numbers for the values in each group. That is, the function calculates the value according to the formula: Sequence number = **(rank - 1) / (totalrows - 1)**. **rank** is the corresponding sequence number generated based on the RANK function for the value and **totalrows** is the total number of elements in a group.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE percent_rank_t1(a int, b int);

gaussdb=# INSERT INTO percent_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,PERCENT_RANK() OVER(PARTITION BY a ORDER BY b) FROM percent_rank_t1;
a | b | percent_rank
-----+-----
1 | 1 | 0
1 | 1 | 0
1 | 2 | .6666666666666667
1 | 3 | 1
2 | 4 | 0
2 | 5 | 1
3 | 6 | 0
```

(7 rows)

```
gaussdb=# DROP TABLE percent_rank_t1;
```

- CUME\_DIST()

Description: The CUME\_DIST function is used for generating accumulative distribution sequence numbers for the values in each group. That is, the function calculates the value according to the following formula: Sequence number = Number of rows preceding or peer with current row/Total rows.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE cume_dist_t1(a int, b int);
```

```
gaussdb=# INSERT INTO cume_dist_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);
```

```
gaussdb=# SELECT a,b,CUME_DIST() OVER(PARTITION BY a ORDER BY b) FROM cume_dist_t1;
```

```
a | b | cume_dist
```

```

```

```
1 | 1 | .5
1 | 1 | .5
1 | 2 | .75
1 | 3 | 1
2 | 4 | .5
2 | 5 | 1
3 | 6 | 1
```

(7 rows)

```
gaussdb=# DROP TABLE cume_dist_t1;
```

- NTILE(num\_buckets integer)

Description: The NTILE function is used for equally allocating sequential data sets to the buckets whose quantity is specified by **num\_buckets** according to **num\_buckets integer** and allocating the bucket number to each row. Divide the partition as evenly as possible.

Return type: integer

Example:

```
gaussdb=# CREATE TABLE ntile_t1(a int, b int);
```

```
gaussdb=# INSERT INTO ntile_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);
```

```
gaussdb=# SELECT a,b,NTILE(2) OVER(PARTITION BY a ORDER BY b) FROM ntile_t1;
```

```
a | b | ntile
```

```

```

```
1 | 1 | 1
1 | 1 | 1
1 | 2 | 2
1 | 3 | 2
2 | 4 | 1
2 | 5 | 2
3 | 6 | 1
```

(7 rows)

```
gaussdb=# DROP TABLE ntile_t1;
```

- LAG(value any [, offset integer [, default any ]])

Description: The LAG function is used for generating lag values for the corresponding values in each group. That is, the value of the row obtained by moving forward the row corresponding to the current value by **offset** (integer) is the sequence number. If the row does not exist after the moving, the result value is the default value. If omitted, **offset** defaults to **1** and **default** to **NULL**. The type of the **default** value must be the same as that of the **value** value.

Return type: same as the parameter type

Example:

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
gaussdb=# INSERT INTO ta1 VALUES('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('24-JUL-05', 'Tobias', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('24-DEC-05', 'Baida', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('18-MAY-03', 'Khoo', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('15-NOV-06', 'Himuro', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-07', 'yq', 11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-08', 'zi', 11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(' ', 'yq1', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, 'yq2', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-DEC-07', 'yq3', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null, 11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null, 11);
INSERT 0 1

-- Call LAG and set offset to 3 and default to null.
gaussdb=# SELECT hire_date, last_name, department_id, lag(hire_date, 3, null) OVER (PARTITION BY
department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY department_id;
 hire_date | last_name | department_id | NextHired
-----+-----+-----+-----
2007-05-10 00:00:00 | yq | 11 |
2008-05-10 00:00:00 | zi | 11 |
 | | 11 |
 | | 11 | 2007-05-10 00:00:00
2005-12-24 00:00:00 | Baida | 30 |
2007-08-10 00:00:00 | Colmenares | 30 |
2006-11-15 00:00:00 | Himuro | 30 |
2003-05-18 00:00:00 | Khoo | 30 | 2005-12-24 00:00:00
2002-12-07 00:00:00 | Raphaely | 30 | 2007-08-10 00:00:00
2005-07-24 00:00:00 | Tobias | 30 | 2006-11-15 00:00:00
 | yq1 | 30 | 2003-05-18 00:00:00
 | yq2 | 30 | 2002-12-07 00:00:00
2007-12-10 00:00:00 | yq3 | 30 | 2005-07-24 00:00:00
(13 rows)
```

- LEAD(value any [, offset integer [, default any ]])

Description: The LEAD function is used for generating leading values for the corresponding values in each group. That is, the value of the row obtained by moving backward the row corresponding to the current value by **offset** (integer) is the sequence number. If the row after the moving exceeds the total number of rows for the current group, the result value is the default value. If omitted, **offset** defaults to **1** and **default** to **NULL**. The type of the **default** value must be the same as that of the **value** value.

Return type: same as the parameter type

Example:

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
```

```

CREATE TABLE
gaussdb=# INSERT INTO ta1 values('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('24-JUL-05', 'Tobias', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('24-DEC-05', 'Baida', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('18-MAY-03', 'Khoo', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('15-NOV-06', 'Himuro', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-07', 'yq', 11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-08', 'zi', 11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('', 'yq1', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, 'yq2', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-DEC-07', 'yq3', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null, 11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null, 11);
INSERT 0 1

-- Call LEAD and set offset to 2.
gaussdb=# SELECT hire_date, last_name, department_id, lead(hire_date, 2) OVER (PARTITION BY
department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY department_id;
 hire_date | last_name | department_id | NextHired
-----+-----+-----+-----
2007-05-10 00:00:00 | yq | 11 |
2008-05-10 00:00:00 | zi | 11 |
 | | 11 |
 | | 11 |
2005-12-24 00:00:00 | Baida | 30 | 2006-11-15 00:00:00
2007-08-10 00:00:00 | Colmenares | 30 | 2003-05-18 00:00:00
2006-11-15 00:00:00 | Himuro | 30 | 2002-12-07 00:00:00
2003-05-18 00:00:00 | Khoo | 30 | 2005-07-24 00:00:00
2002-12-07 00:00:00 | Raphaely | 30 |
2005-07-24 00:00:00 | Tobias | 30 |
 | yq1 | 30 | 2007-12-10 00:00:00
 | yq2 | 30 |
2007-12-10 00:00:00 | yq3 | 30 |
(13 rows)

```

- **FIRST\_VALUE(value any)**

Description: Returns the first value of each group.

Return type: same as the parameter type

Example:

```

gaussdb=# CREATE TABLE first_value_t1(a int, b int);

gaussdb=# INSERT INTO first_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,FIRST_VALUE(b) OVER(PARTITION BY a ORDER BY b) FROM first_value_t1;
 a | b | first_value
-----+-----+-----
 1 | 1 | 1
 1 | 1 | 1
 1 | 2 | 1
 1 | 3 | 1
 2 | 4 | 4
 2 | 5 | 4
 3 | 6 | 6
(7 rows)

```

```
gaussdb=# DROP TABLE first_value_t1;
```

- **LAST\_VALUE(value any)**

Description: Returns the last value of each group.

Return type: same as the parameter type

Example:

```
gaussdb=# CREATE TABLE last_value_t1(a int, b int);
```

```
gaussdb=# INSERT INTO last_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);
```

```
gaussdb=# SELECT a,b, LAST_VALUE(b) OVER(PARTITION BY a ORDER BY b) FROM last_value_t1;
```

```
a | b | last_value
```

```

```

```
1 | 1 | 1
```

```
1 | 1 | 1
```

```
1 | 2 | 2
```

```
1 | 3 | 3
```

```
2 | 4 | 4
```

```
2 | 5 | 5
```

```
3 | 6 | 6
```

```
(7 rows)
```

```
gaussdb=# DROP TABLE last_value_t1;
```

- **NTH\_VALUE(value any, nth integer)**

Description: The *n*th row for a group is the returned value. If the row does not exist, **NULL** is returned by default.

Return type: same as the parameter type

Example:

```
gaussdb=# CREATE TABLE nth_value_t1(a int, b int);
```

```
gaussdb=# INSERT INTO nth_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);
```

```
gaussdb=# SELECT a,b,NTH_VALUE(b, 2) OVER(PARTITION BY a order by b) FROM nth_value_t1;
```

```
a | b | nth_value
```

```

```

```
1 | 1 | 1
```

```
1 | 1 | 1
```

```
1 | 2 | 1
```

```
1 | 3 | 1
```

```
2 | 4 | 5
```

```
2 | 5 | 5
```

```
3 | 6 |
```

```
(7 rows)
```

```
gaussdb=# DROP TABLE nth_value_t1;
```

- **delta**

Description: Returns the difference between the current row and the previous row.

Parameter: numeric

Return type: numeric

- **spread**

Description: Calculates the difference between the maximum value and minimum value in a certain period.

Parameter: real

Return type: real

## 7.5.20 Security Functions

### Security Functions

- `gs_encrypt_aes128(encryptstr,keystr)`  
Description: Encrypts **encryptstr** strings using the key derived from **keystr** and returns encrypted strings. The value of **keystr** ranges from 8 to 16 bytes and contains at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters. Currently, the following types of data can be encrypted: numerals supported in the database; character type; RAW in binary type; and DATE, TIMESTAMP, and SMALLDATETIME in date/time type.

Return type: text

Length of the return value: At least 92 bytes and no more than  $(4*[Len/3]+68)$  bytes, where *Len* indicates the length of the data before encryption (unit: byte).

Example:

```
gaussdb=# SELECT gs_encrypt_aes128('MPPDB','1234@abc');
 gs_encrypt_aes128
```

```

OF1g3+70oeqFfyKiWlpxfYxPnpeitNc6+7nAe02Tt37fZF8Q+bbEYhdw/YG+0c9tHKRWM6OcTzLB3HnqvX
+1d8Bflo=
(1 row)
```

#### NOTE

A password is required during the execution of this function. For security purposes, the `gsq` tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in `gsq` by paging up and down.

- `gs_decrypt_aes128(decryptstr,keystr)`

Description: Decrypts **decrypt** strings using the key derived from **keystr** and returns decrypted strings. The **keystr** used for decryption must be consistent with that used for encryption. **keystr** cannot be empty.

#### NOTE

This function needs to be used with the `gs_encrypt_aes128` encryption function.

Return type: text

Example:

```
gaussdb=# SELECT gs_decrypt_aes128('OF1g3+70oeqFfyKiWlpxfYxPnpeitNc6+7nAe02Tt37fZF8Q
+bbEYhdw/YG+0c9tHKRWM6OcTzLB3HnqvX+1d8Bflo=', '1234@abc');
 gs_decrypt_aes128
```

```

MPPDB
(1 row)
```

#### NOTE

A password is required during the execution of this function. For security purposes, the `gsq` tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in `gsq` by paging up and down.

- `aes_encrypt(str, key_str, init_vector)`

Description: Encrypts the string **str** using the key string **key\_str** and initialization vector **init\_vector** based on the AES algorithm.

Parameters in the command above are as follows:

- **str**: character string to be encrypted. If **str** is set to **NULL**, the function returns **NULL**.
- **key\_str**: key character string. If **key\_str** is set to **NULL**, the function returns **NULL**. For security purposes, you are advised to use a 128-bit, 192-bit, or 256-bit secure random number as the key character string if the key length is 128 bits, 192 bits, or 256 bits (determined by the value of **block\_encryption\_mode**).
- **init\_vector**: An initialization variable is provided for the required block encryption mode. The length is greater than or equal to 16 bytes. Bytes greater than 16 bytes are automatically ignored. If neither **str** nor **key\_str** is **NULL**, this parameter cannot be **NULL**. Otherwise, an error is reported. For security purposes, you are advised to ensure that the IV for each encryption is unique in OFB mode and that the IV for each encryption is unpredictable in CBC or CFB mode.

Return type: text

Example:

```
gaussdb=# SELECT aes_encrypt('huwei123','123456vfhex4dyu,vdaladhjsadad','1234567890123456');
aes_encrypt

u*8\x05c?0
(1 row)
```

#### NOTE

- This function is valid only when GaussDB is compatible with MySQL (that is, **sql\_compatibility** is set to 'MYSQL').
  - A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.
  - Do not call this function during operations related to stored procedures, preventing the risk of sensitive information disclosure. In addition, when using the stored procedure that contains the function, you are advised to filter the parameter information of the function before providing the information for external maintenance personnel to locate the fault. Delete the logs after using them.
  - Do not call the function when **debug\_print\_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the function in the log files generated when **debug\_print\_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.
  - The **SQL\_ASCII** setting is different from other settings. If the character set of the server is **SQL\_ASCII**, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If this parameter is set to **SQL\_ASCII**, no code conversion occurs. When this function calls the third-party OpenSSL library, the returned data is non-ASCII data. Therefore, when the character set of the database server is set to **SQL\_ASCII**, the encoding of the client must also be set to **SQL\_ASCII**. Otherwise, an error is reported. The database does not convert or verify non-ASCII characters.
- **aes\_decrypt(pass\_str, key\_str, init\_vector)**  
Description: Decrypts the string **str** using the key string **key\_str** and initialization vector **init\_vector** based on the AES algorithm.  
Parameters in the command above are as follows:
    - **pass\_str**: character string to be decrypted. If **pass\_str** is set to **NULL**, the function returns **NULL**.

- **key\_str**: key character string. If **key\_str** is set to **NULL**, the function returns **NULL**. For security purposes, you are advised to use a 128-bit, 192-bit, or 256-bit secure random number as the key character string if the key length is 128 bits, 192 bits, or 256 bits (determined by the value of **block\_encryption\_mode**).
- **init\_vector**: An initialization variable is provided for the required block decryption mode. The length is greater than or equal to 16 bytes. Bytes greater than 16 bytes are automatically ignored. If neither **pass\_str** nor **key\_str** is **NULL**, this parameter cannot be **NULL**. Otherwise, an error is reported. For security purposes, you are advised to ensure that the IV for each encryption is unique in OFB mode and that the IV for each encryption is unpredictable in CBC or CFB mode.

Return type: text

Example:

```
gaussdb=# SELECT
aes_decrypt(aes_encrypt('huwei123','123456vfhex4dyu,vdaladhjsadad','1234567890123456'),'123456vf
hex4dyu,vdaladhjsadad','1234567890123456');
aes_decrypt

huwei123
(1 row)
```

#### NOTE

- This function is valid only when GaussDB is compatible with MySQL (that is, **sql\_compatibility** is set to 'MYSQL').
  - A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.
  - Do not call this function during operations related to stored procedures, preventing the risk of sensitive information disclosure. In addition, when using the stored procedure that contains the function, you are advised to filter the parameter information of the function before providing the information for external maintenance personnel to locate the fault. Delete the logs after using them.
  - Do not call the function when **debug\_print\_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the function in the log files generated when **debug\_print\_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.
  - To ensure successful decryption, ensure that the values of **block\_encryption\_mode**, **key\_str** and IV are the same as those during encryption.
  - Due to encoding differences, encrypted data cannot be directly copied from the gsql client for decryption. In this scenario, the decryption result may not be the character string before encryption.
  - The SQL\_ASCII setting is different from other settings. If the character set of the server is SQL\_ASCII, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If this parameter is set to SQL\_ASCII, no code conversion occurs. When this function calls the third-party OpenSSL library, the returned data is non-ASCII data. Therefore, when the character set of the database server is set to SQL\_ASCII, the encoding of the client must also be set to SQL\_ASCII. Otherwise, an error is reported. The database does not convert or verify non-ASCII characters.
- **gs\_digest(input\_string, hash\_algorithm)**

Description: Hashes the input string using the specified hash algorithm and returns a hexadecimal number.

Parameters in the command above are as follows:

- **input\_string**: character string to be hashed. The value cannot be **NULL**.
- **hash\_algorithm**: specifies the hash algorithm. Currently, SHA-256, SHA-384, SHA-512, and SM3 are supported. Both uppercase and lowercase letters are supported. If an unsupported hash algorithm is used, an error is reported.

Return type: text

Example:

```
gaussdb=# SELECT pg_catalog.gs_digest('gaussdb', 'sha256');
 gs_digest

4dc50d746f4e04f9b446986b34a0050e358fbfb8bc1fba314c54b52a417b0b8e
(1 row)
```

- **gs\_password\_deadline**

Description: Indicates the number of remaining days before the password of the current user expires.

Return type: interval

Example:

```
gaussdb=# SELECT gs_password_deadline();
 gs_password_deadline

83 days 17:44:32.196094
(1 row)
```

- **gs\_password\_notifytime()**

Description: Specifies the number of days prior to password expiration that a user will receive a reminder.

Return type: int32

- **login\_audit\_messages(BOOLEAN)**

Description: Queries login information about a login user.

Return type: tuple

Example:

- Check the date, time, and IP address of the last successful login.

```
gaussdb=> SELECT * FROM login_audit_messages(true);
username | database | logintime | mytype | result | client_conninfo
-----+-----+-----+-----+-----+-----
omm | postgres | 2020-06-29 21:56:40+08 | login_success | ok | gsq!@[local]
(1 row)
```

- Check the number, date, and time of failed attempts since the previous successful login.

```
gaussdb=> SELECT * FROM login_audit_messages(false);
username | database | logintime | mytype | result | client_conninfo
-----+-----+-----+-----+-----+-----
omm | postgres | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local]
omm | postgres | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local]
(2 rows)
```

- **login\_audit\_messages\_pid**

Description: Queries login information about a login user. Different from **login\_audit\_messages**, this function queries login information based on **backendid**. Information about subsequent logins of the same user does not alter the query result of previous logins and cannot be found using this function.

Return type: tuple

 NOTE

When the thread pool is enabled, the **backendid** obtained in the same session may change due to thread switchover. As a result, the return values are different when the function is called for multiple times. You are advised not to call this function when the thread pool is enabled.

Example:

- Check the date, time, and IP address of the last successful login.

```
gaussdb=> SELECT * FROM login_audit_messages_pid(true);
username | database | logintime | mytype | result | client_conninfo | backendid
-----+-----+-----+-----+-----+-----+-----
omm | postgres | 2020-06-29 21:56:40+08 | login_success | ok | gsqr@[local] | 139823109633792
(1 row)
```

- Check the number, date, and time of failed attempts since the previous successful login.

```
gaussdb=> SELECT * FROM login_audit_messages_pid(false);
username | database | logintime | mytype | result | client_conninfo | backendid
-----+-----+-----+-----+-----+-----+-----
omm | postgres | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local] | 139823109633792
omm | postgres | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local] | 139823109633792
(2 rows)
```

- **inet\_server\_addr**

Description: Displays the server IP address.

Return type: inet

Example:

```
gaussdb=# SELECT inet_server_addr();
inet_server_addr

10.10.0.13
(1 row)
```

 NOTE

- The client IP address 10.10.0.50 and server IP address 10.10.0.13 are used as an example.
- If the database is connected to the local PC, the value is empty.

- **inet\_client\_addr**

Description: Displays the client IP address.

Return type: inet

Example:

```
gaussdb=# SELECT inet_client_addr();
inet_client_addr

10.10.0.50
(1 row)
```

 NOTE

- The client IP address 10.10.0.50 and server IP address 10.10.0.13 are used as an example.
- If the database is connected to the local PC, the value is empty.

- **pg\_query\_audit**

Description: Displays audit logs of the current CN.

Return type: record

The following table describes return fields.

| Name            | Type                     | Description                                 |
|-----------------|--------------------------|---------------------------------------------|
| time            | timestamp with time zone | Operation time                              |
| type            | text                     | Operation                                   |
| result          | text                     | Operation result                            |
| userid          | oid                      | User ID                                     |
| username        | text                     | Name of the user who performs the operation |
| database        | text                     | Database name                               |
| client_conninfo | text                     | Client connection information               |
| object_name     | text                     | Object name                                 |
| detail_info     | text                     | Operation details                           |
| node_name       | text                     | Node name                                   |
| thread_id       | text                     | Thread ID                                   |
| local_port      | text                     | Local port                                  |
| remote_port     | text                     | Remote port                                 |

- **pgxc\_query\_audit**  
Description: Displays audit logs of all CNs.  
Return type: record  
The return fields of this function are the same as those of the `pg_query_audit` function.
- **pg\_delete\_audit**  
Description: Deletes audit logs in a specified period.  
Return type: void
- **alldigitsmasking**  
Description: Specifies the internal function of the masking policy, which is used to anonymize all characters.  
Parameter: col text, letter character default '0'  
Return type: text
- **creditcardmasking**  
Description: Specifies the internal function of the masking policy, which is used to anonymize all credit card information.  
Parameter: col text, letter character default 'x'  
Return type: text

- **randommasking**  
Description: Specifies the internal function of the masking policy. The random policy is used.  
Parameter: col text  
Return type: text
- **fullemailmasking**  
Description: Specifies the internal function of the masking policy, which is used to anonymize the text before the last period (.) except the at sign (@).  
Parameter: col text, letter character default 'x'  
Return type: text
- **basicemailmasking**  
Description: Specifies the internal function of the masking policy, which is used to anonymize the text before the first at sign (@).  
Parameter: col text, letter character default 'x'  
Return type: text
- **shufflemasking**  
Description: Specifies the internal function of the masking policy, which is used to sort characters out of order.  
Parameter: col text  
Return type: text
- **regexprmasking**  
Description: Specifies the internal function of the masking policy, which is used to replace characters using a regular expression.  
Parameter: col text, reg text, replace\_text text, pos INTEGER default 0, reg\_len INTEGER default -1  
Return type: text
- **gs\_encrypt(encryptstr,keystr, encrypttype)**  
Description: Encrypts **encryptstr** strings using **keystr** as the encryption password and returns encrypted strings based on **encrypttype**. The value of **keystr** contains 8 to 16 bytes and at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters. The value of **encrypttype** can be **aes128** or **sm4**.  
Return type: text  
Example:

```
gaussdb=# SELECT gs_encrypt('MPPDB','Asdf1234','sm4');
gs_encrypt

ZBzOmaGA4Bb+coyucJ0B8AkIShqc
(1 row)
```
- **gs\_decrypt(decryptstr,keystr,decrypttype)**

 **NOTE**

An encryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.

Description: Decrypts **decrypt** strings using **keyst** as the decryption password and returns decrypted strings based on **decrypttype**. **decrypttype** and **keyst** used for decryption must be consistent with **encrypttype** and **keyst** used for encryption. The value of **keyst** cannot be empty. The value of **decrypttype** can be **aes128** or **sm4**.

This function needs to be used with the `gs_encrypt` encryption function.

Return type: text

Example:

```
gaussdb=# SELECT gs_decrypt('ZBzOmaGA4Bb+coyucJ0B8AkiShqc','Asdf1234','sm4');
gs_decrypt

MPPDB
(1 row)
```

#### NOTE

A decryption password is required during the execution of this function. For security purposes, the `gsq` tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in `gsq` by paging up and down.

## 7.5.21 Encrypted Functions and Operators

- `byteawithoutorderwithequalcolin(cstring)`  
Description: Converts input data to the internal `byteawithoutorderwithequalcol` format.  
Parameter type: `cstring`  
Return type: `byteawithoutorderwithequalcol`
- `byteawithoutorderwithequalcolout(byteawithoutorderwithequalcol)`  
Description: Converts internal data of the `byteawithoutorderwithequalcol` type to data of the `cstring` type.  
Parameter type: `byteawithoutorderwithequalcol`  
Return type: `cstring`
- `byteawithoutorderwithequalcolsend(byteawithoutorderwithequalcol)`  
Description: Converts data of the `byteawithoutorderwithequalcol` type to data of the `bytea` type.  
Parameter type: `byteawithoutorderwithequalcol`  
Return type: `bytea`
- `byteawithoutorderwithequalcolrecv(internal)`  
Description: Converts data of the internal type to data of the `byteawithoutorderwithequalcol` type.  
Parameter type: `internal`  
Return type: `byteawithoutorderwithequalcol`
- `byteawithoutorderwithequalcoltypmodin(cstring[])`  
Description: Converts data of the `cstring[]` type to data of the `byteawithoutorderwithequalcol` type.  
Parameter type: `cstring[]`  
Return type: `int4`

- `byteawithoutorderwithequalcoltypmodout(int4)`  
Description: Converts data of the `int4` type into data of the `cstring` type.  
Parameter type: `int4`  
Return type: `cstring`
- `byteawithoutordercolin(cstring)`  
Description: Converts input data to the internal `byteawithoutordercolin` format.  
Parameter type: `cstring`  
Return type: `byteawithoutordercol`
- `byteawithoutordercolout(byteawithoutordercol)`  
Description: Converts internal data of the `byteawithoutordercol` type to data of the `cstring` type.  
Parameter type: `byteawithoutordercol`  
Return type: `cstring`
- `byteawithoutordercolsend(byteawithoutordercol)`  
Description: Converts data of the `byteawithoutordercol` type to data of the `bytea` type.  
Parameter type: `byteawithoutordercol`  
Return type: `bytea`
- `byteawithoutordercolrecv(internal)`  
Description: Converts data of the `internal` type to data of the `byteawithoutordercol` type.  
Parameter type: `internal`  
Return type: `byteawithoutordercol`
- `byteawithoutorderwithequalcolcmp(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)`  
Description: Compares two `byteawithoutorderwithequalcol` data sizes. If the first data size is smaller than the second one, `-1` is returned. If the first data size is equal to the second one, `0` is returned. If the first data size is larger than the second one, `1` is returned.  
Parameter type: `byteawithoutorderwithequalcol, byteawithoutorderwithequalcol`  
Return type: `int4`
- `byteawithoutorderwithequalcolcmpbytear(byteawithoutorderwithequalcol, bytea)`  
Description: Compares the `byteawithoutorderwithequalcol` and `bytea` data sizes. If the first data size is smaller than the second one, `-1` is returned. If the first data size is equal to the second one, `0` is returned. If the first data size is larger than the second one, `1` is returned.  
Parameter type: `byteawithoutorderwithequalcol` or `bytea`  
Return type: `int4`
- `byteawithoutorderwithequalcolcmpbyteal(bytea, byteawithoutorderwithequalcol)`  
Description: Compares the `bytea` and `byteawithoutorderwithequalcol` data sizes. If the first data size is smaller than the second one, `-1` is returned. If the

first data size is equal to the second one, **0** is returned. If the first data size is larger than the second one, **1** is returned.

Parameter type: bytea, byteawithoutorderwithequalcol

Return type: int4

- `byteawithoutorderwithequalcoleq(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)`

Description: Compares two `byteawithoutorderwithequalcol` data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

Parameter type: `byteawithoutorderwithequalcol`, `byteawithoutorderwithequalcol`

Return type: Boolean

- `byteawithoutorderwithequalcoleqbytea(bytea, byteawithoutorderwithequalcol)`

Description: Compares the `bytea` and `byteawithoutorderwithequalcol` data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

Parameter type: `bytea`, `byteawithoutorderwithequalcol`

Return type: Boolean

- `byteawithoutorderwithequalcoleqbytea(byteawithoutorderwithequalcol, bytea)`

Description: Compares the `byteawithoutorderwithequalcol` and `bytea` data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

Parameter type: `byteawithoutorderwithequalcol`, `bytea`

Return type: Boolean

- `byteawithoutorderwithequalcolne(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)`

Description: Compares two `byteawithoutorderwithequalcol` data records. If they are different, **true** is returned. Otherwise, **false** is returned.

Parameter type: `byteawithoutorderwithequalcol`, `byteawithoutorderwithequalcol`

Return type: Boolean

- `byteawithoutorderwithequalcolnebytea(bytea, byteawithoutorderwithequalcol)`

Description: Compares the `bytea` and `byteawithoutorderwithequalcol` data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

Parameter type: `bytea`, `byteawithoutorderwithequalcol`

Return type: Boolean

- `byteawithoutorderwithequalcolnebytea(byteawithoutorderwithequalcol, bytea)`

Description: Compares the `byteawithoutorderwithequalcol` and `bytea` data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

Parameter type: `byteawithoutorderwithequalcol`, `bytea`

Return type: Boolean

- `hll_hash_byteawithoutorderwithequalcol(byteawithoutorderwithequalcol)`

Description: Returns the hll hash value of `byteawithoutorderwithequalcol`.

Parameter type: byteawithoutorderwithequalcol  
Return type: hll\_hashval

## Example

Encrypted equality functions such as `byteawithoutorderwithequalcolin` and `byteawithoutorderwithequalcolout` are read/write format conversion functions such as `in`, `out`, `send`, and `recv` specified by the data type `byteawithoutorderwithequalcol` in the database kernel. For details, see the `byteain` and `byteaout` functions of the `bytea` type. However, the local CEK is verified, and the function can be successfully executed only when the encrypted column contains a `cekoid` that exists on the local host.

```
-- For example, if the int_type encrypted table exists, int_col2 is the encrypted column.
-- Use a non-encrypted client to connect to the database and query the ciphertext of the encrypted column.
gaussdb=# SELECT int_col2 FROM int_type;
 int_col2

\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424
919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6
(1 row)

-- The ciphertext of the encrypted column is used as the input parameter of
byteawithoutorderwithequalcolin. The format is converted from cstring to byteawithoutorderwithequalcol.
gaussdb=# SELECT
byteawithoutorderwithequalcolin('\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a
6ac7371acc0ac33100000035fe3424919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6');
 byteawithoutorderwithequalcolin

\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424
919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6
(1 row)
```

Implementations such as `byteawithoutorderwithequalcolin` search for CEK and determine whether it is a normal encrypted data type.

Therefore, if the format of the data entered by the user is not the encrypted data format and the corresponding CEK does not exist on the local host, an error is returned.

```
gaussdb=# SELECT * FROM
byteawithoutorderwithequalcolsend('\x907219912381298461289346129':byteawithoutorderwithequalcol);
ERROR: cek with OID 596711794 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('\x907219912...
 ^

gaussdb=# SELECT * FROM byteawithoutordercolout('\x9072190199999999999912381298461289346129');
ERROR: cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutordercolout('\x907219019999999999...

gaussdb=# SELECT * FROM
byteawithoutorderwithequalcolrecv('\x9072190199999999999912381298461289346129':byteawithoutorde
rwithequalcol);
ERROR: cek with OID 2566986098 not found
 ^

gaussdb=# SELECT * FROM
byteawithoutorderwithequalcolsend('\x9072190199999999999912381298461289346129':byteawithoutorde
rwithequalcol);
ERROR: cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('\x907219019...
 ^
```

## 7.5.22 Set Returning Functions

### Series Generating Functions

- `generate_series(start, stop)`  
 Description: Generates a series of values, from **start** to **stop** with a step size of one.  
 Parameter type: int, bigint, numeric  
 Return type: setof int, setof bigint, setof numeric (same as the parameter type)
- `generate_series(start, stop, step)`  
 Description: Generates a series of values, from **start** to **stop** with a step size of **step**.  
 Parameter type: int, bigint, numeric  
 Return type: setof int, setof bigint, setof numeric (same as the parameter type)
- `generate_series(start, stop, step interval)`  
 Description: Generates a series of values, from **start** to **stop** with a step size of **step**.  
 Parameter type: timestamp or timestamp with time zone  
 Return type: setof timestamp or setof timestamp with time zone (same as parameter type)

When **step** is positive, zero rows are returned if **start** is greater than **stop**. Conversely, when **step** is negative, zero rows are returned if **start** is less than **stop**. Zero rows are also returned for **NULL** inputs. It is an error for **step** to be zero.

Example:

```
gaussdb=# SELECT * FROM generate_series(2,4);
generate_series

 2
 3
 4
(3 rows)

gaussdb=# SELECT * FROM generate_series(5,1,-2);
generate_series

 5
 3
 1
(3 rows)

gaussdb=# SELECT * FROM generate_series(4,3);
generate_series

(0 rows)

-- This example applies to the date-plus-integer operator.
gaussdb=# SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS s(a);
dates

2017-06-02
2017-06-09
2017-06-16
```

```
(3 rows)
gaussdb=# SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10 hours');
 generate_series

2008-03-01 00:00:00
2008-03-01 10:00:00
2008-03-01 20:00:00
2008-03-02 06:00:00
2008-03-02 16:00:00
2008-03-03 02:00:00
2008-03-03 12:00:00
2008-03-03 22:00:00
2008-03-04 08:00:00
(9 rows)
```

## Subscript Generating Functions

- `generate_subscripts(array anyarray, dim int)`  
Description: Generates a series comprising the given array's subscripts.  
Return type: setof int
- `generate_subscripts(array anyarray, dim int, reverse boolean)`  
Description: Generates a series comprising the given array's subscripts. When **reverse** is true, the series is returned in reverse order.  
Return type: setof int

**generate\_subscripts** is a function that generates the set of valid subscripts for the specified dimension of the given array. Zero rows are returned for arrays that do not have the requested dimension, or for NULL arrays (but valid subscripts are returned for NULL array elements). Example:

```
-- Basic usage
gaussdb=# SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s

 1
 2
 3
 4
(4 rows)
-- Unnest a 2D array.
gaussdb=# CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
SELECT $1[i][j]
FROM generate_subscripts($1,1) g1(i),
generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;

gaussdb=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
unnest2

 1
 2
 3
 4
(4 rows)

-- Delete the function.
gaussdb=# DROP FUNCTION unnest2;
```

## 7.5.23 Conditional Expression Functions

### Conditional Expression Functions

- `coalesce(expr1, expr2, ..., exprn)`

Description:

Returns the first of its arguments that are not null.

**COALESCE(expr1, expr2)** is equivalent to **CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END**.

Example:

```
gaussdb=# SELECT coalesce(NULL,'hello');
coalesce

hello
(1 row)
```

Note:

- Null is returned only if all parameters are null.
- This value is replaced by the default value when data is displayed.
- Like a **CASE** expression, **COALESCE** only evaluates the parameters that are needed to determine the result. That is, parameters to the right of the first non-NULL parameter are not evaluated.

- `decode(base_expr, compare1, value1, Compare2,value2, ... default)`

Description: Compares **base\_expr** with each **compare(n)** and **returns value(n)** if they are matched. If no match occurs, default is returned.

Example:

```
gaussdb=# SELECT decode('A','A',1,'B',2,0);
case

1
(1 row)
```

Note: Operations on the XML data are not supported.

- `nullif(expr1, expr2)`

Description: Returns **NULL** only when **expr1** is equal to **expr2**. Otherwise, **expr1** is returned.

**nullif(expr1, expr2)** is equivalent to **CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END**.

Example:

```
gaussdb=# SELECT nullif('hello','world');
nullif

hello
(1 row)
```

Note: Operations on the XML data are not supported.

Assume the two parameter data types are different:

- If implicit conversion exists between the two data types, implicitly convert the parameter of lower priority to this data type using the data type of higher priority. If the conversion succeeds, computation is performed.

Otherwise, an error is reported. Example:

```
gaussdb=# SELECT nullif('1234'::VARCHAR,123::INT4);
nullif
```

```

1234
(1 row)
gaussdb=# SELECT nullif('1234'::VARCHAR,'2012-12-24'::DATE);
ERROR: invalid input syntax for type timestamp: "1234"
```

- If implicit conversion is not applied between two data types, an error is displayed. Example:

```
gaussdb=# SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE);
ERROR: operator does not exist: boolean = timestamp without time zone
LINE 1: SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE) FROM sys_dummy,
^
HINT: No operator matches the given name and argument type(s). You might need to add explicit type casts.
```

- `nvl( expr1 , expr2 )`

Description:

- If the value of **expr1** is **NULL**, the value of **expr2** is returned.
- If the value of **expr1** is not **NULL**, the value of **expr1** is returned.

Example:

```
gaussdb=# SELECT nvl('hello','world');
nvl

hello
(1 row)
```

Parameters **expr1** and **expr2** can be of any data type. If **expr1** and **expr2** are of different data types, NVL checks whether **expr2** can be implicitly converted to **expr1**. If it can, the **expr1** data type is returned. If **expr2** cannot be implicitly converted to **expr1** but **expr1** can be implicitly converted to **expr2**, the **expr2** data type is returned. If no implicit type conversion exists between the two parameters and the parameters are different data types, an error is reported.

- `nvl2(expr1, expr2, expr3)`

Description:

- If *expr1* is **NULL**, *expr3* is returned.
- If *expr1* is not **NULL**, *expr2* is returned.

#### NOTE

This function is valid only when the value of **a\_format\_version** is **10c** and that of **a\_format\_dev\_version** is **s1** in an ORA-compatible database.

Example:

```
gaussdb=# SELECT nvl2('hello','world','other');
case

world
(1 row)
```

Note: The *expr2* and *expr3* parameters can be of any type. If the last two parameters of NVL2 are of different types, check whether *expr3* can be implicitly converted to *expr2*. If *expr3* cannot be implicitly converted to *expr2*, an error is returned. If the first parameter is of the numeric type, the function converts this parameter and other parameters to the numeric type, and then compares them. If the parameters cannot be converted, an error message is displayed. If the first parameter is of another type, the function converts other parameters to the type of the first parameter for comparison. If the parameters cannot be converted, an error message is displayed.

- `greatest(expr1 [, ...])`

Description: Selects the largest value from a list of any number of expressions.

Return type:

Example:

```
gaussdb=# SELECT greatest(1*2,2-3,4-1);
greatest

 3
(1 row)
gaussdb=# SELECT greatest('HARRY', 'HARRIOT', 'HAROLD');
greatest

HARRY
(1 row)
```

Note: Operations on the XML data are not supported.

#### NOTE

In the scenario where this function is in an ORA-compatible database, the value of **a\_format\_version** is **10c**, and that of **a\_format\_dev\_version** is **s1**:

1. If the value of any parameter is null, the function returns null.
2. If the first parameter is of the numeric type, the function converts this parameter and other parameters to the numeric type, and then compares them. If the parameters cannot be converted, an error message is displayed. If the first parameter is of another type, the function converts other parameters to the type of the first parameter for comparison. If the parameters cannot be converted, an error message is displayed.

- **least(expr1 [, ...])**

Description: Selects the smallest value from a list of any number of expressions.

Example:

```
gaussdb=# SELECT least(1*2,2-3,4-1);
least

 -1
(1 row)
gaussdb=# SELECT least('HARRY','HARRIOT','HAROLD');
least

HAROLD
(1 row)
```

Note: Operations on the XML data are not supported.

#### NOTE

In the scenario where this function is in an ORA-compatible database, the value of **a\_format\_version** is **10c**, and that of **a\_format\_dev\_version** is **s1**:

1. If the value of any parameter is null, the function returns null.
2. If the first parameter is of the numeric type, the function converts this parameter and other parameters to the numeric type, and then compares them. If the parameters cannot be converted, an error message is displayed. If the first parameter is of another type, the function converts other parameters to the type of the first parameter for comparison. If the parameters cannot be converted, an error message is displayed.

- **EMPTY\_BLOB()**

Description: Initiates a BLOB variable in an **INSERT** or an **UPDATE** statement to a **NULL** value.

Return type: BLOB

## Example:

```
-- Create a table.
gaussdb=# CREATE TABLE blob_tb(b blob,id int) DISTRIBUTE BY REPLICATION;
-- Insert data.
gaussdb=# INSERT INTO blob_tb VALUES (empty_blob(),1);
-- Drop the table.
gaussdb=# DROP TABLE blob_tb;
```

Note: The length is 0 obtained using DBE\_LOB.GET\_LENGTH.

- EMPTY\_CLOB()

Description: Initiates a CLOB variable in an INSERT or UPDATE statement to a null value.

 NOTE

This function is valid only when the value of **a\_format\_version** is **10c** and that of **a\_format\_dev\_version** is **s1** in an ORA-compatible database.

Return type: CLOB

## Example:

```
-- Create a table.
gaussdb=# CREATE TABLE clob_tb(c clob,id int);
-- Insert data.
gaussdb=# INSERT INTO clob_tb VALUES (empty_clob(),1);
-- Drop the table.
gaussdb=# DROP TABLE clob_tb;
```

Note: The length is 0 obtained using DBE\_LOB.GET\_LENGTH.

- Lnnvl(condition)

Description: Checks the condition in the WHERE clause of a query statement. If the condition is true, **false** is returned. If the condition is unknown or false, **true** is returned.

**condition:** The value must be a logical expression. However, it cannot be used in composite conditions with keywords such as AND, OR, and BETWEEN.

Return type: Boolean

## Example:

```
-- Create a table.
gaussdb=# CREATE TABLE student_demo (name VARCHAR2(20), grade NUMBER(10,2));
CREATE TABLE

-- Insert data.
gaussdb=# INSERT INTO student_demo VALUES ('name0',0);
INSERT 0 1
gaussdb=# INSERT INTO student_demo VALUES ('name1',1);
INSERT 0 1
gaussdb=# INSERT INTO student_demo VALUES ('name2',2);
INSERT 0 1

-- Call Lnnvl.
gaussdb=# SELECT * FROM student_demo WHERE LNNVL(name = 'name1');
 name | grade
-----+-----
name0 | 0.00
name2 | 2.00
(2 rows)

-- Drop the table.
gaussdb=# drop table student_demo;
DROP TABLE
```

 NOTE

The Innvl function is supported only when the value of **a\_format\_version** is **10c** and the value of **a\_format\_dev\_version** is **s2** in an ORA-compatible database.

## 7.5.24 System Information Functions

### Session Information Functions

- **SYS\_CONTEXT()**

Description: Returns the value of the parameter associated with the context namespace at the current time.

Return type: text

Example:

```
select SYS_CONTEXT('userenv','NLS_CURRENCY');
sys_context

$
(1 row)

select SYS_CONTEXT('userenv','NLS_DATE_FORMAT');
sys_context

ISO, MDY
(1 row)

select SYS_CONTEXT('userenv','NLS_DATE_LANGUAGE');
sys_context

en_US.UTF-8
(1 row)
```

- **current\_catalog**

Description: Name of the current database (called "catalog" in the SQL standard)

Return type: name

Example:

```
testdb=# SELECT current_catalog;
current_database

testdb
(1 row)
```

- **current\_database()**

Description: Name of the current database

Return type: name

Example:

```
testdb=# SELECT current_database();
current_database

testdb
(1 row)
```

- **current\_query()**

Description: Text of the currently executing query committed by the client (which might contain more than one statement)

Return type: text

Example:

```
gaussdb=# SELECT current_query();
current_query

SELECT current_query();
(1 row)
```

- `current_schema[()]`

Description: Name of the current schema

Return type: name

Example:

```
gaussdb=# SELECT current_schema();
current_schema

public
(1 row)
```

Note: **current\_schema** returns the first valid schema name in the search path. (If the search path is empty or contains no valid schema name, **NULL** is returned.) This is the schema that will be used for any tables or other named objects that are created without specifying a target schema.

- `current_schemas(Boolean)`

Description: Name of a schema in the search path

Return type: name[]

Example:

```
gaussdb=# SELECT current_schemas(true);
current_schemas

{pg_catalog,public}
(1 row)
```

Note:

**current\_schemas(Boolean)** returns an array of the names of all schemas in the search path. The Boolean option specifies whether implicitly included system schemas such as **pg\_catalog** are included in the returned search path.

#### NOTE

The search path can be altered at the run time. The command is as follows:

```
SET search_path TO schema [, schema, ...]
```

- `current_user`

Description: Username in the current operating environment.

Return type: name

Example:

```
gaussdb=# SELECT current_user;
current_user

omm
(1 row)
```

Note: **current\_user** is the user identifier used for permission check. Normally it is equal to the session user, but it can be changed by using **SET ROLE**. It also changes during the execution of functions with the **SECURITY DEFINER** attribute.

- `definer_current_user`

Description: Username in the current operating environment.

Return type: name

Example:

```
gaussdb=# SELECT definer_current_user();
definer_current_user

omm
(1 row)
```

Note: In most cases, the results of **definer\_current\_user** and **current\_user** are the same. However, when this function is executed in a stored procedure, the name of user who defines the current stored procedure is returned.

- **pg\_current\_sessionid()**

Description: Session ID of the current execution context.

Return type: text

Example:

```
gaussdb=# SELECT pg_current_sessionid();
pg_current_sessionid

1579228402.140190434944768
(1 row)
```

Note: **pg\_current\_sessionid()** is used to obtain the session ID in the current operating environment. The format of the value is *Timestamp.Session ID*. When **enable\_thread\_pool** is set to **off**, the actual session ID is the thread ID.

- **pg\_current\_sessid**

Description: Session ID of the current execution context.

Return type: text

Example:

```
gaussdb=# select pg_current_sessid();
pg_current_sessid

140308875015936
(1 row)
```

Note: In thread pool mode, the session ID of the current session is obtained. In non-thread pool mode, the backend thread ID of the current session is obtained.

- **pg\_current\_userid**

Description: Current user ID

Return type: text

Example:

```
gaussdb=# SELECT pg_current_userid();
pg_current_userid

10
(1 row)
```

- **tablespace\_oid\_name()**

Description: Queries the tablespace name based on the tablespace OID.

Return type: text

Example:

```
gaussdb=# select tablespace_oid_name(1663);
tablespace_oid_name

pg_default
(1 row)
```

- `inet_client_addr()`

Description: Remote connection address. **inet\_client\_addr** returns the IP address of the current client.

 **NOTE**

It is available only in remote connection mode.

Return type: `inet`

Example:

```
gaussdb=# SELECT inet_client_addr();
inet_client_addr

10.10.0.50
(1 row)
```

- `inet_client_port()`

Description: Remote connection port. **inet\_client\_port** returns the port number of the current client.

 **NOTE**

It is available only in remote connection mode.

Return type: `int`

Example:

```
gaussdb=# SELECT inet_client_port();
inet_client_port

33143
(1 row)
```

- `inet_server_addr()`

Description: Local connection address. **inet\_server\_addr** returns the IP address on which the server accepts the current connection.

 **NOTE**

It is available only in remote connection mode.

Return type: `inet`

Example:

```
gaussdb=# SELECT inet_server_addr();
inet_server_addr

10.10.0.13
(1 row)
```

- `inet_server_port()`

Description: Local connection port. **inet\_server\_port** returns the number of the port receiving the current connection. All these functions return **NULL** if the current connection is via a UDS.

 **NOTE**

It is available only in remote connection mode.

Return type: `int`

Example:

```
gaussdb=# SELECT inet_server_port();
inet_server_port

```

```
8000
(1 row)
```

- `pg_backend_pid()`

Description: Process ID of the service process attached to the current session.

Return type: int

Example:

```
gaussdb=# SELECT pg_backend_pid();
 pg_backend_pid

140229352617744
(1 row)
```

- `pg_conf_load_time()`

Description: Configures load time. **pg\_conf\_load\_time** returns the timestamp when the server configuration files were last loaded.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT pg_conf_load_time();
 pg_conf_load_time

2017-09-01 16:05:23.89868+08
(1 row)
```

- `pg_my_temp_schema()`

Description: OID of the temporary schema of a session. The value is **0** if the OID does not exist.

Return type: oid

Example:

```
gaussdb=# SELECT pg_my_temp_schema();
 pg_my_temp_schema

0
(1 row)
```

Note: **pg\_my\_temp\_schema** returns the OID of the current session's temporary schema, or **0** if it has no temporary schemas (because no temporary tables are created). **pg\_is\_other\_temp\_schema** returns **true** if the given OID is the OID of another session's temporary schema.

- `pg_is_other_temp_schema(oid)`

Description: Specifies whether the schema is the temporary schema of another session.

Return type: Boolean

Example:

```
gaussdb=# SELECT pg_is_other_temp_schema(25356);
 pg_is_other_temp_schema

f
(1 row)
```

- `pg_listening_channels()`

Description: Name of the channel that the session is currently listening to.

Return type: SETOF text

Example:

```
gaussdb=# SELECT pg_listening_channels();
 pg_listening_channels
```

```

(0 rows)
```

Note: **pg\_listening\_channels** returns a set of names of channels that the current session is currently listening to.

- **pg\_postmaster\_start\_time()**

Description: Server start time. **pg\_postmaster\_start\_time** returns the **timestamp with time zone** when the server is started.

Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT pg_postmaster_start_time();
pg_postmaster_start_time

2017-08-30 16:02:54.99854+08
(1 row)
```

- **sessionid2pid()**

Description: Obtains PID information from a session ID (for example, the **sessid** column in **pv\_session\_stat**).

Return type: int8

Example:

```
gaussdb=# select sessionid2pid(sessid::cstring) from pv_session_stat limit 2;
sessionid2pid

139973107902208
139973107902208
(2 rows)
```

- **session\_context('namespace', 'parameter')**

Description: Obtains and returns the parameter values of a specified namespace.

Return type: VARCHAR

Example:

```
gaussdb=# SELECT session_context('USERENV', 'CURRENT_SCHEMA');
session_context

public
(1 row)
```

The result varies according to the current actual schema.

Note: Currently, only the **SESSION\_CONTEXT('USERENV', 'CURRENT\_SCHEMA')** and **SESSION\_CONTEXT('USERENV', 'CURRENT\_USER')** formats are supported.

- **pg\_trigger\_depth()**

Description: Nesting level of triggers.

Return type: int

Example:

```
gaussdb=# SELECT pg_trigger_depth();
pg_trigger_depth

0
(1 row)
```

- **opengauss\_version()**

Description: openGauss version information.

Return type: text

The following is an example. Replace *x.x.x* in the query result with the actual value.

```
gaussdb=# SELECT opengauss_version();
opengauss_version

x.x.x
(1 row)
```

- `gs_deployment()`

Description: Deployment mode of the current system. For a distributed system, **Distribute** is returned.

Return type: text

Example:

```
gaussdb=# select gs_deployment();
gs_deployment

Distribute
(1 row)
```

- `session_user`

Description: Session username.

Return type: name

Example:

```
gaussdb=# SELECT session_user;
session_user

omm
(1 row)
```

Note: **session\_user** usually specifies the initial user connected to the current database, but system administrators can change this setting by using **SET SESSION AUTHORIZATION**.

- `user`

Description: Equivalent to `current_user`.

Return type: name

Example:

```
gaussdb=# SELECT user;
current_user

omm
(1 row)
```

- `get_shard_oids_byname`

Description: Returns the OID of the node when the node name is entered.

Return type: oid

Example:

```
gaussdb=# select get_shard_oids_byname('datanode1');
get_shard_oids_byname

{16385}
(1 row)
```

- `getpgusername()`

Description: Obtains the database username.

Return type: name

Example:

```
gaussdb=# select getpgusername();
getpgusername

GaussDB_userna
(1 row)
```

- `getdatabaseencoding()`

Description: Obtains the database encoding mode.

Return type: name

Example:

```
gaussdb=# select getdatabaseencoding();
getdatabaseencoding

SQL_ASCII
(1 row)
```

- `version()`

Description: version information. **version** returns a string describing a server's version.

Return type: text

Example:

```
gaussdb=# SELECT version();
 version

gaussdb (GaussDB 503.1.XXX build f521c606) compiled at 2021-09-16 14:55:22 commit 2935 last mr
6385 release
(1 row)
```

- `working_version_num()`

Description: version number. It returns a version number related to system compatibility.

Return type: int

Example:

```
gaussdb=# SELECT working_version_num();
working_version_num

 92231
(1 row)
```

- `get_hostname()`

Description: Returns the host name of the current node.

Return type: text

Example:

```
gaussdb=# SELECT get_hostname();
get_hostname

linux-user
(1 row)
```

- `get_nodename()`

Description: Returns the name of the current node.

Return type: text

Example:

```
gaussdb=# SELECT get_nodename();
get_nodename

```

```
coordinator1
(1 row)
```

- `get_schema_oid(cstring)`

Description: Returns the OID of the queried schema.

Return type: oid

Example:

```
gaussdb=# SELECT get_schema_oid('public');
get_schema_oid

 2200
(1 row)
```

- `pgxc_parse_clog(OUT xid int8, OUT nodename text, OUT status text)`

Description: Returns the status of all transactions in the current cluster.

Return type: SETOF record

Example:

```
gaussdb=# SELECT pgxc_parse_clog();
pgxc_parse_clog

(0,dn_6004_6005_6006,INPROGRESS)
(1,dn_6004_6005_6006,COMMITTED)
(2,dn_6004_6005_6006,INPROGRESS)
(3 row)
```

- `pgxc_prepared_xact( )`

Description: Returns the list of transaction GIDs at the prepared stage in the cluster.

Return type: set of text

Example:

```
gaussdb=# SELECT pgxc_prepared_xact();
pgxc_prepared_xact

(0 row)
```

- `pgxc_xacts_iscommitted()`

Description: Returns the status of the transaction with the specified XID in the cluster. **t** indicates the committed state, **f** indicates the aborted state, and **null** indicates other states. Users must have the SYSADMIN or MONADMIN permission.

Return type: SETOF record

Example:

```
gaussdb=# SELECT pgxc_xacts_iscommitted(1);
pgxc_xacts_iscommitted

(dn_6004_6005_6006,t)
(cn_5001,t)
(cn_5002,t)
(dn_6001_6002_6003,t)
(4 row)
```

- `pgxc_total_memory_detail()`

Description: Displays the memory usage in the cluster. Users must have the SYSADMIN or MONADMIN permission.

#### NOTE

If the GUC parameter `enable_memory_limit` is set to `off`, this function cannot be used.

Return type: set of pv\_total\_memory\_detail

Example:

```
gaussdb=# SELECT pgxc_total_memory_detail();
pgxc_total_memory_detail

(dn_6004_6005_6006,max_process_memory,81920)
(dn_6004_6005_6006,process_used_memory,72747)
(dn_6004_6005_6006,max_dynamic_memory,12096)
(dn_6004_6005_6006,dynamic_used_memory,1530)
(4 row)
```

- pv\_total\_memory\_detail

Description: Collects statistics on memory usage of the current database node in the unit of MB.

 **NOTE**

If the GUC parameter **enable\_memory\_limit** is set to **off**, this function cannot be used.

Return type: record

**Table 7-41** Return value description

| Name     | Type | Description |
|----------|------|-------------|
| nodename | text | Node name   |

| Name        | Type    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| memorytype  | text    | <p>Memory type. The value must be one of the following:</p> <ul style="list-style-type: none"> <li>• <b>max_process_memory</b>: memory occupied by a GaussDB cluster instance</li> <li>• <b>process_used_memory</b>: memory occupied by a GaussDB process</li> <li>• <b>max_dynamic_memory</b>: maximum dynamic memory</li> <li>• <b>dynamic_used_memory</b>: used dynamic memory</li> <li>• <b>dynamic_peak_memory</b>: dynamic peak memory</li> <li>• <b>dynamic_used_shrctx</b>: maximum dynamic shared memory context</li> <li>• <b>dynamic_peak_shrctx</b>: dynamic peak value of the shared memory context</li> <li>• <b>max_shared_memory</b>: maximum shared memory</li> <li>• <b>shared_used_memory</b>: used shared memory</li> <li>• <b>max_sctpcomm_memory</b>: maximum memory allowed for the communications library</li> <li>• <b>sctpcomm_used_memory</b>: memory used by the communications library</li> <li>• <b>sctpcomm_peak_memory</b>: memory peak of the communications library</li> <li>• <b>other_used_memory</b>: other used memory</li> </ul> |
| memorybytes | integer | Size of allocated memory-typed memory                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

- `get_client_info()`  
Description: Returns client information.  
Return type: record

## Access Permission Query Functions

The DDL permissions, including ALTER, DROP, COMMENT, INDEX, and VACUUM, are inherent permissions implicitly owned by the owner.

The following access permission query functions only specify whether a user has a certain permission on an object (that is, a permission on an object recorded in the **acl** column of a system catalog is returned):

- `has_any_column_privilege(user, table, privilege)`

Description: Queries whether a specified user has permissions on any column of a table.

**Table 7-42** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------|----------------------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user      | name, oid                  | Users       | Username or ID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| table     | text, oid                  | Tables      | Table name or ID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| privilege | text                       | Permission  | <ul style="list-style-type: none"> <li>● <b>SELECT</b>: allows the <b>SELECT</b> statement to be executed on any column of a specified table.</li> <li>● <b>INSERT</b>: allows the <b>INSERT</b> statement to be executed on any column of a specified table.</li> <li>● <b>UPDATE</b>: allows the <b>UPDATE</b> statement to be executed on any column of a specified table.</li> <li>● <b>REFERENCES</b>: allows a foreign key constraint (not supported in distributed mode).</li> <li>● <b>COMMENT</b>: allows the <b>COMMENT</b> statement to be executed on any column of a specified table.</li> </ul> |

Return type: Boolean

- `has_any_column_privilege(table, privilege)`

Description: Queries whether the current user has the permission to access any column of a table. For details about the valid parameter types, see [Table 7-42](#).

Return type: Boolean

Note: **has\_any\_column\_privilege** checks whether a user can access any column of a table in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**, except that the desired access permission type must be some combination of **SELECT**, **INSERT**, **UPDATE**, or **REFERENCES**.

 **NOTE**

Note that having any of these permissions at the table level indicates that the permission is implicitly granted for each column of the table. Therefore, **has\_any\_column\_privilege** always returns **true** if **has\_table\_privilege** has the same parameters. A success message is also returned if a column-level permission is granted for at least one column.

- `has_column_privilege(user, table, column, privilege)`  
Description: Specifies whether a specified user has the permission to access columns.

**Table 7-43** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|----------------------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user      | name, oid                  | Users       | Username or ID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| table     | text, oid                  | Table name  | Table name or ID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| column    | text, smallint             | Column name | Name or attribute number of a column                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| privilege | text                       | Permission  | <ul style="list-style-type: none"> <li>• <b>SELECT</b>: allows the <b>SELECT</b> statement to be executed on specified columns of a table.</li> <li>• <b>INSERT</b>: allows the <b>INSERT</b> statement to be executed on specified columns of a table.</li> <li>• <b>UPDATE</b>: allows the <b>UPDATE</b> statement to be executed on specified columns of a table.</li> <li>• <b>REFERENCES</b>: allows a foreign key constraint (not supported in distributed mode).</li> <li>• <b>COMMENT</b>: allows the <b>COMMENT</b> statement to be executed on specified columns of a table.</li> </ul> |

Return type: Boolean

- `has_column_privilege(table, column, privilege)`  
Description: Specifies whether the current user has the permission to access columns. For details about the valid parameter types, see [Table 7-43](#).

Return type: Boolean

**has\_column\_privilege** checks whether a user can access a column in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**. Columns can be added either by name or by attribute number. The desired access permission type must be some combination of **SELECT**, **INSERT**, **UPDATE**, or **REFERENCES**.

 **NOTE**

Note that having any of these permissions at the table level indicates that the permission is implicitly granted for each column of the table.

- `has_cek_privilege(user, cek, privilege)`  
Description: Specifies whether a specified user has permissions on CEKs.

**Table 7-44** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                |
|-----------|----------------------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user      | name, oid                  | Users       | Username or ID                                                                                                                                                             |
| cek       | text, oid                  | CEK         | Name or ID of a CEK                                                                                                                                                        |
| privilege | text                       | Permission  | <ul style="list-style-type: none"> <li>• <b>USAGE</b>: allows users to use the specified CEK.</li> <li>• <b>DROP</b>: allows users to delete the specified CEK.</li> </ul> |

Return type: Boolean

- `has_cmk_privilege(user, cmk, privilege)`  
Description: Specifies whether a specified user has permissions on CMKs.

**Table 7-45** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                |
|-----------|----------------------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user      | name, oid                  | Users       | Username or ID                                                                                                                                                             |
| cmk       | text, oid                  | CMK         | Name or ID of the CMK                                                                                                                                                      |
| privilege | text                       | Permission  | <ul style="list-style-type: none"> <li>• <b>USAGE</b>: allows users to use the specified CMK.</li> <li>• <b>DROP</b>: allows users to delete the specified CMK.</li> </ul> |

Return type: Boolean

- `has_database_privilege(user, database, privilege)`  
Description: Specifies whether a specified user has permissions on databases.

**Table 7-46** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|----------------------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user      | name, oid                  | Users       | Username or ID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| database  | text, oid                  | Database    | Database name or ID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| privilege | text                       | Permission  | <ul style="list-style-type: none"> <li>● <b>CREATE</b>: For databases, allows new schemas to be created within the database.</li> <li>● <b>TEMPORARY</b>: allows users to create temporary tables when the database is used.</li> <li>● <b>TEMP</b>: allows users to create temporary tables when the database is used.</li> <li>● <b>CONNECT</b>: allows users to access specified databases.</li> <li>● <b>ALTER</b>: allows users to modify the attributes of a specified object.</li> <li>● <b>DROP</b>: allows users to delete a specified object.</li> <li>● <b>COMMENT</b>: allows users to define or modify comments of a specified object.</li> </ul> |

Return type: Boolean

- has\_database\_privilege(database, privilege)

Description: Queries whether the current user has the permission to access a database. For details about the valid parameter types, see [Table 7-46](#).

Return type: Boolean

Note: **has\_database\_privilege** checks whether a user can access a database in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**. The desired access permission type must be some combination of **CREATE**, **CONNECT**, **TEMPORARY**, or **TEMP** (which is equivalent to **TEMPORARY**).

- has\_directory\_privilege(user, directory, privilege)

**Table 7-47** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                  |
|-----------|----------------------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user      | name, oid                  | Users       | Username or ID                                                                                                                                                               |
| directory | text, oid                  | Directory   | Directory name or OID                                                                                                                                                        |
| privilege | text                       | Permission  | <ul style="list-style-type: none"> <li>• <b>READ</b>: allows read operations on the directory.</li> <li>• <b>WRITE</b>: allows write operations on the directory.</li> </ul> |

Description: Specifies whether a specified user has permissions on directories.

Return type: Boolean

- `has_directory_privilege(directory, privilege)`

Description: Queries whether the current user has the permission to access a directory. For details about the valid parameter types, see [Table 7-47](#).

Return type: Boolean

- `has_foreign_data_wrapper_privilege(user, fdw, privilege)`

**Table 7-48** Parameter type description

| Parameter | Valid Input Parameter Type | Description          | Value Range                                               |
|-----------|----------------------------|----------------------|-----------------------------------------------------------|
| user      | name, oid                  | Users                | Username or ID                                            |
| fdw       | text, oid                  | Foreign data wrapper | Name or ID of the foreign data wrapper                    |
| privilege | text                       | Permission           | <b>USAGE</b> : allows access to the foreign data wrapper. |

Description: Specifies whether a specified user has permissions on foreign data wrappers.

Return type: Boolean

- `has_foreign_data_wrapper_privilege(fdw, privilege)`

Description: Queries whether the current user has permissions on foreign data wrappers. For details about the valid parameter types, see [Table 7-48](#).

Return type: Boolean

Note: **has\_foreign\_data\_wrapper\_privilege** checks whether a user can access a foreign data wrapper in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**. The desired access permission type must be **USAGE**.

- `has_function_privilege(user, function, privilege)`

**Table 7-49** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------|----------------------------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user      | name, oid                  | Users       | Username or ID                                                                                                                                                                                                                                                                                                                                                                                                     |
| function  | text, oid                  | Function    | Function name or ID                                                                                                                                                                                                                                                                                                                                                                                                |
| privilege | text                       | Permission  | <p><b>EXECUTE</b>: allows users to use specified functions and the operators that are realized by the functions.</p> <ul style="list-style-type: none"> <li>• <b>ALTER</b>: allows users to modify the attributes of a specified object.</li> <li>• <b>DROP</b>: allows users to delete a specified object.</li> <li>• <b>COMMENT</b>: allows users to define or modify comments of a specified object.</li> </ul> |

Description: Specifies whether a specified user has permissions on functions.

Return type: Boolean

- `has_function_privilege(function, privilege)`

Description: Specifies whether the current user has permissions on functions. For details about valid parameter types, see [Table 7-49](#).

Return type: Boolean

Note: **has\_function\_privilege** checks whether a user can access a function in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**. When a function is specified by a text string rather than by an OID, the allowed input is the same as that for the **regprocedure** data type (see [Object Identifier Types](#)). The desired access permission type must be **EXECUTE**.

- `has_language_privilege(user, language, privilege)`

**Table 7-50** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range    |
|-----------|----------------------------|-------------|----------------|
| user      | name, oid                  | Users       | Username or ID |

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                           |
|-----------|----------------------------|-------------|---------------------------------------------------------------------------------------|
| language  | text, oid                  | Language    | Language name or ID                                                                   |
| privilege | text                       | Permission  | <b>USAGE:</b> allows users to specify a procedural language when creating a function. |

Description: Specifies whether a specified user has permissions on languages.

Return type: Boolean

- has\_language\_privilege(language, privilege)

Description: Specifies whether the current user has permissions on languages. For details about valid parameter types, see [Table 7-50](#).

Return type: Boolean

Note: **has\_language\_privilege** checks whether a user can access a procedural language in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**. The desired access permission type must be **USAGE**.

- has\_nodegroup\_privilege(user, nodegroup, privilege)

Description: Checks whether a user has permission to access a cluster node.

Return type: Boolean

**Table 7-51** Parameter type description

| Parameter | Valid Input Parameter Type | Description  | Value Range             |
|-----------|----------------------------|--------------|-------------------------|
| user      | name, oid                  | Users        | Existing username or ID |
| nodegroup | text, oid                  | Cluster node | Existing cluster node   |

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|----------------------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| privilege | text                       | Permission  | <ul style="list-style-type: none"> <li>• <b>USAGE</b>: For sub-clusters, allows users who can access objects contained in the schema to access tables in the sub-cluster.</li> <li>• <b>CREATE</b>: For sub-clusters, allows users to create tables within the sub-cluster.</li> <li>• <b>COMPUTE</b>: allows elastic computing in the sub-cluster.</li> <li>• <b>ALTER</b>: allows users to modify the attributes of a specified object.</li> <li>• <b>DROP</b>: allows users to delete a specified object.</li> </ul> |

- `has_nodegroup_privilege(nodegroup, privilege)`  
Description: Checks whether a user has permission to access a cluster node.  
Return type: Boolean
- `has_schema_privilege(user, schema, privilege)`  
Description: Specifies whether a specified user has permissions on schemas.  
Return type: Boolean
- `has_schema_privilege(schema, privilege)`  
Description: Specifies whether the current user has permissions on schemas.  
Return type: Boolean  
Note: **has\_schema\_privilege** checks whether a user can access a schema in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**. The desired access permission type must be **CREATE**, **USAGE**, **ALTER**, **DROP**, or **COMMENT**.
- `has_sequence_privilege(user, sequence, privilege)`  
Description: Queries whether a specified user has permissions on sequences.  
Return type: Boolean

**Table 7-52** Parameter type description

| Parameter | Valid Input Parameter Type | Description | Value Range                  |
|-----------|----------------------------|-------------|------------------------------|
| user      | name, oid                  | Users       | Existing username or ID      |
| sequence  | text, oid                  | Sequence    | Existing sequence name or ID |

| Parameter | Valid Input Parameter Type | Description | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------|----------------------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| privilege | text                       | Permission  | <ul style="list-style-type: none"> <li>• <b>USAGE</b>: For sequences, allows users to use the <b>nextval</b> function.</li> <li>• <b>SELECT</b>: allows users to create a sequence.</li> <li>• <b>UPDATE</b>: allows users to execute the <b>UPDATE</b> statement.</li> <li>• <b>ALTER</b>: allows users to modify the attributes of a specified object.</li> <li>• <b>DROP</b>: allows users to delete a specified object.</li> <li>• <b>COMMENT</b>: allows users to define or modify comments of a specified object.</li> </ul> |

- `has_sequence_privilege(sequence, privilege)`  
Description: Queries whether the current user has permissions on sequences.  
Return type: Boolean
- `has_server_privilege(user, server, privilege)`  
Description: Specifies whether a specified user has permissions on foreign servers.  
Return type: Boolean
- `has_server_privilege(server, privilege)`  
Description: Specifies whether the current user has permissions on foreign servers.  
Return type: Boolean  
Note: **has\_server\_privilege** checks whether a user can access a foreign server in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**. The access permission type must be **USAGE**, **ALTER**, **DROP**, or **COMMENT**.
- `has_table_privilege(user, table, privilege)`  
Description: Specifies whether a specified user has permissions on tables.  
Return type: Boolean
- `has_table_privilege(table, privilege)`  
Description: Specifies whether the current user has permissions on tables.  
Return type: Boolean  
Note: **has\_table\_privilege** checks whether a user can access a table in a particular way. The user can be specified by name or by OID (**pg\_authid.oid**), or be set to **public** which indicates the PUBLIC role. If this parameter is omitted, **current\_user** is used. The table can be specified by name or by OID.

When it is specified by name, the name can be schema-qualified if necessary. If the desired access permission type is specified by a text string, the text string must be one of the values **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **TRUNCATE**, **REFERENCETRIGGER**, **ALTER**, **DROP**, **COMMENT**, **INDEX**, or **VACUUM**. Optionally, **WITH GRANT OPTION** can be added to a permission type to test whether the permission is held with the grant option. Also, multiple permission types can be listed separated by commas, in which case the result will be **true** if any of the listed permissions is held.

Example:

```
gaussdb=# CREATE TABLE tt(a int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# SELECT has_table_privilege('tt', 'select');
 has_table_privilege

t
(1 row)

View the database username list.
gaussdb=# \du;
 List of roles
Role name | Attributes | Member of
-----+-----+-----
omm | Sysadmin, Create role, Create DB, Replication, Administer audit, Monitoradmin, Operatoradmin, Policyadmin, UseFT | {}
simple_user | | {}
test | Sysadmin | {}

gaussdb=# SELECT has_table_privilege('omm', 'tt', 'select,INSERT WITH GRANT OPTION ');
 has_table_privilege

t
(1 row)

gaussdb=# DROP TABLE tt;
DROP TABLE
```

- **has\_tablespace\_privilege(user, tablespace, privilege)**  
Description: Specifies whether a specified user has permissions on tablespaces.  
Return type: Boolean
- **has\_tablespace\_privilege(tablespace, privilege)**  
Description: Specifies whether the current user has permissions on tablespaces.  
Return type: Boolean  
Note: **has\_tablespace\_privilege** checks whether a user can access a tablespace in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**. The access permission type must be **CREATE**, **ALTER**, **DROP**, or **COMMENT**.
- **pg\_has\_role(user, role, privilege)**  
Description: Specifies whether a specified user has permissions on roles.  
Return type: Boolean
- **pg\_has\_role(role, privilege)**  
Description: Specifies whether the current user has permissions on roles.  
Return type: Boolean

Note: **pg\_has\_role** checks whether a user can access a role in a particular way. Its parameter possibilities are analogous to those of **has\_table\_privilege**, except that **public** cannot be used as a username. The desired access permission type must be some combination of **MEMBER** or **USAGE**. **MEMBER** denotes direct or indirect membership in the role (that is, permission **SET ROLE**), while **USAGE** denotes the usage permission on the role that is available without the **SET ROLE** permission.

- **has\_any\_privilege**(user, privilege)

Description: Queries whether a specified user has certain ANY permission. If multiple permissions are queried at the same time, **true** is returned as long as one permission is obtained.

Return type: Boolean

**Table 7-53** Parameter type description

| Parameter | Valid Input Parameter Type | Description    | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------|----------------------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user      | name                       | Users          | Existing username                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| privilege | text                       | ANY permission | Available values: <ul style="list-style-type: none"> <li>• CREATE ANY TABLE [WITH ADMIN OPTION]</li> <li>• ALTER ANY TABLE [WITH ADMIN OPTION]</li> <li>• DROP ANY TABLE [WITH ADMIN OPTION]</li> <li>• SELECT ANY TABLE [WITH ADMIN OPTION]</li> <li>• INSERT ANY TABLE [WITH ADMIN OPTION]</li> <li>• UPDATE ANY TABLE [WITH ADMIN OPTION]</li> <li>• DELETE ANY TABLE [WITH ADMIN OPTION]</li> <li>• CREATE ANY SEQUENCE [WITH ADMIN OPTION]</li> <li>• CREATE ANY INDEX [WITH ADMIN OPTION]</li> <li>• CREATE ANY FUNCTION [WITH ADMIN OPTION]</li> <li>• EXECUTE ANY FUNCTION [WITH ADMIN OPTION]</li> <li>• CREATE ANY TYPE [WITH ADMIN OPTION]</li> </ul> |

## Schema Visibility Query Functions

Each function performs the visibility check on one type of database objects. For functions and operators, an object in the search path is visible if there is no object of the same name and parameter data type earlier in the path. For operator classes, both name and associated index access methods are considered.

All these functions require OIDs to identify the objects to be checked. If you want to test an object by name, it is convenient to use the OID alias type (**regclass**, **regtype**, **regprocedure**, **regoperator**, **regconfig**, or **regdictionary**).

For example, a table is said to be visible if the schema where the table is located is in the search path and no table of the same name appears earlier in the search path. This is equivalent to the statement that the table can be referenced by name without explicit schema qualification. To list the names of all visible tables, run the following command:

```
gaussdb=# SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

- **pg\_collation\_is\_visible(collation\_oid)**  
Description: Specifies whether the collation is visible in the search path.  
Return type: Boolean
- **pg\_conversion\_is\_visible(conversion\_oid)**  
Description: Specifies whether the conversion is visible in the search path.  
Return type: Boolean
- **pg\_function\_is\_visible(function\_oid)**  
Description: Specifies whether the function is visible in the search path.  
Return type: Boolean
- **pg\_opclass\_is\_visible(opclass\_oid)**  
Description: Specifies whether the operator class is visible in the search path.  
Return type: Boolean
- **pg\_operator\_is\_visible(operator\_oid)**  
Description: Specifies whether the operator is visible in the search path.  
Return type: Boolean
- **pg\_opfamily\_is\_visible(opclass\_oid)**  
Description: Specifies whether the operator family is visible in the search path.  
Return type: Boolean
- **pg\_table\_is\_visible(table\_oid)**  
Description: Specifies whether the table is visible in the search path.  
Return type: Boolean
- **pg\_ts\_config\_is\_visible(config\_oid)**  
Description: Specifies whether the text search configuration is visible in the search path.  
Return type: Boolean
- **pg\_ts\_dict\_is\_visible(dict\_oid)**  
Description: Specifies whether the text search dictionary is visible in the search path.

- Return type: Boolean

  - `pg_ts_parser_is_visible(parser_oid)`  
Description: Specifies whether the text search parser is visible in the search path.  
Return type: Boolean
  - `pg_ts_template_is_visible(template_oid)`  
Description: Specifies whether the text search template is visible in the search path.  
Return type: Boolean
  - `pg_type_is_visible(type_oid)`  
Description: Specifies whether the type (or domain) is visible in the search path.  
Return type: Boolean

## System Catalog Information Functions

- `format_type(type_oid, typemod)`  
Description: Obtains the SQL name of a data type.  
Return type: text  
Note: **format\_type** returns the SQL name of a data type based on the OID of the data type and possible modifiers. If the specific modifier is unknown, pass **NULL** at the position of the modifier. Modifiers are generally meaningful only for data types with length restrictions. The SQL name returned by **format\_type** contains the length of the data type, which can be calculated by taking `sizeof(int32)` from actual storage length [actual storage len - `sizeof(int32)`] in the unit of bytes. 32-bit space is required to store the customized length set by users. Therefore, the actual storage length contains 4 bytes more than the customized length. In the following example, the SQL name returned by **format\_type** is `character varying(6)`, indicating the length of the `varchar` type is 6 bytes. Therefore, the actual storage length of the `varchar` type is 10 bytes.

```
gaussdb=# SELECT format_type((SELECT oid FROM pg_type WHERE typename='varchar'), 10);
format_type

character varying(6)
(1 row)
```
- `pg_check_authid(role_oid)`  
Description: Checks whether a role name with given OID exists.  
Return type: Boolean

```
gaussdb=# select pg_check_authid(1);
pg_check_authid

f
(1 row)
```
  - `pg_describe_object(catalog_id, object_id, object_sub_id)`  
Description: Obtains the description of a database object.  
Return type: text  
Note: **pg\_describe\_object** returns the description of a database object specified by a catalog OID, an object OID, and a (possibly zero) sub-object ID.

This is useful to determine the identity of an object stored in the **pg\_depend** catalog.

- **pg\_get\_constraintdef(constraint\_oid)**  
Description: Obtains definition of a constraint.  
Return type: text
- **pg\_get\_constraintdef(constraint\_oid, pretty\_bool)**  
Description: Obtains definition of a constraint.  
Return type: text  
Note: **pg\_get\_constraintdef** and **pg\_get\_indexdef** respectively reconstruct the creation command for a constraint and an index.
- **pg\_get\_expr(pg\_node\_tree, relation\_oid)**  
Description: Decompiles the internal form of an expression, assuming that any Vars in it refer to the relationship indicated by the second parameter.  
Return type: text
- **pg\_get\_expr(pg\_node\_tree, relation\_oid, pretty\_bool)**  
Description: Decompiles the internal form of an expression, assuming that any Vars in it refer to the relationship indicated by the second parameter.  
Return type: text  
Note: **pg\_get\_expr** decompiles the internal form of an individual expression, such as the default value of a column. This helps to check the content of system catalogs. If the expression might contain keywords, specify the OID of the relationship they refer to as the second parameter; if no keywords are expected, zero is sufficient.
- **pg\_get\_functiondef(func\_oid)**  
Description: Obtains the definition of a function.  
Return type: text

Example:

```
gaussdb=# SELECT * FROM pg_get_functiondef(598);
headerlines | definition
-----+-----
4 | CREATE OR REPLACE FUNCTION pg_catalog.abbrev(inet)+
 | RETURNS text +
 | LANGUAGE internal +
 | IMMUTABLE STRICT NOT FENCED NOT SHIPPABLE +
 | AS $function$inet_abbrev$function$ +
(1 row)
```

- **pg\_get\_function\_arguments(func\_oid)**  
Description: Obtains argument list of function's definition (with default values).  
Return type: text  
Note: **pg\_get\_function\_arguments** returns the parameter list of a function, in the form it would need to appear in CREATE FUNCTION.
- **pg\_get\_function\_identity\_arguments(func\_oid)**  
Description: Obtains the parameter list to identify a function (without default values).  
Return type: text

Note: `pg_get_function_identity_arguments` returns the parameter list necessary to identify a function, in the form it would need to appear in within `ALTER FUNCTION`. This form omits default values.

- `pg_get_function_result(func_oid)`

Description: Obtains the `RETURNS` clause for a function.

Return type: text

Note: `pg_get_function_result` returns the appropriate `RETURNS` clause for the function.

- `pg_get_indexdef(index_oid)`

Description: Obtains the **CREATE INDEX** command for an index.

Return type: text

Example:

```
gaussdb=# SELECT * FROM pg_get_indexdef(16416);
 pg_get_indexdef

CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
```

- `pg_get_indexdef(index_oid, dump_schema_only)`

Description: Obtains the **CREATE INDEX** command for indexes in dump scenarios. In the current version, the value of **dump\_schema\_only** does not affect the function output.

Return type: text

Example:

```
gaussdb=# SELECT * FROM pg_get_indexdef(16416, true);
 pg_get_indexdef

CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
```

- `pg_get_indexdef(index_oid, column_no, pretty_bool)`

Description: Obtains the **CREATE INDEX** command for an index, or definition of just one index column when the value of **column\_no** is not zero.

Return type: text

Example:

```
gaussdb=# SELECT * FROM pg_get_indexdef(16416, 0, false);
 pg_get_indexdef

CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
gaussdb=# SELECT * FROM pg_get_indexdef(16416, 1, false);
 pg_get_indexdef

b
(1 row)
```

Note: **pg\_get\_functiondef** returns a complete **CREATE OR REPLACE FUNCTION** statement for a function.

- `pg_get_keywords()`

Description: Obtains the list of SQL keywords and their categories.

Return type: SETOF record

Note: **pg\_get\_keywords** returns a set of records describing the SQL keywords recognized by the server. The **word** column contains the keywords. The **catcode** column contains a category code: **U** for unreserved, **C** for column

name, **T** for type or function name, or **R** for reserved. The **catdesc** column contains a possibly-localized string describing the category.

- `pg_get_ruledef(rule_oid)`  
Description: Obtains the **CREATE RULE** command for a rule.  
Return type: text
- `pg_get_ruledef(rule_oid, pretty_bool)`  
Description: Obtains the **CREATE RULE** command for a rule.  
Return type: text
- `pg_get_userbyid(role_oid)`  
Description: Obtains the role name with a given OID.  
Return type: name  
Note: **pg\_get\_userbyid** extracts a role's name given its OID.
- `pg_check_authid(role_id)`  
Description: Checks whether a user exists based on **role\_id**.  
Return type: text  

```
gaussdb=# SELECT pg_check_authid(20);
pg_check_authid

f
(1 row)
```
- `pg_get_viewdef(view_name)`  
Description: Obtains the underlying **SELECT** command for a view.  
Return type: text
- `pg_get_viewdef(view_name, pretty_bool)`  
Description: Obtains the underlying **SELECT** command for a view. Lines with columns are wrapped to 80 columns if **pretty\_bool** is set to **true**.  
Return type: text  
Note: **pg\_get\_viewdef** reconstructs the **SELECT** query that defines a view. Most of these functions come in two forms. When the function has the **pretty\_bool** parameter and the value is **true**, it can optionally "pretty-print" the result. The pretty-printed format is more readable. The other one is the default format which is more likely to be interpreted in the same way by future versions. Avoid using pretty-printed output for dump purposes. Passing **false** to the pretty-print parameter generates the same result as a variant without this parameter.
- `pg_get_viewdef(view_oid)`  
Description: Obtains the underlying **SELECT** command for a view.  
Return type: text
- `pg_get_viewdef(view_oid, pretty_bool)`  
Description: Obtains the underlying **SELECT** command for a view. Lines with columns are wrapped to 80 columns if **pretty\_bool** is set to **true**.  
Return type: text
- `pg_get_viewdef(view_oid, wrap_column_int)`  
Description: Obtains the underlying **SELECT** command for a view. Lines with columns are wrapped to the specified number of columns and printing is implicit.

Return type: text

- `pg_get_tabledef(table_oid)`

Description: Obtains the definition of a table based on **table\_oid**.

Return type: text

Example:

```
gaussdb=# SELECT * FROM pg_get_tabledef(16384);
 pg_get_tabledef

SET search_path = public;
CREATE TABLE t1 (
 c1 bigint DEFAULT nextval('serial'::regclass)
)
WITH (orientation=row, compression=no)
DISTRIBUTE BY HASH(c1)
TO GROUP group1;
(1 row)
```

- `pg_get_tabledef(table_name)`

Description: Obtains the definition of a table based on **table\_name**.

Return type: text

Example:

```
gaussdb=# SELECT * FROM pg_get_tabledef('t1');
 pg_get_tabledef

SET search_path = public;
CREATE TABLE t1 (
 c1 bigint DEFAULT nextval('serial'::regclass)
)
WITH (orientation=row, compression=no)
DISTRIBUTE BY HASH(c1)
TO GROUP group1;
(1 row)
```

Note: **pg\_get\_tabledef** reconstructs the **CREATE** statement of the definition of the table, including the table definition, index information, and comments. Users need to create the dependent objects of the table, such as groups, schemas, tablespaces, and servers. The table definition does not include the statements for creating these dependent objects.

- `pg_options_to_table(reloptions)`

Description: Obtains the set of storage option name/value pairs.

Return type: SETOF record

Note: **pg\_options\_to\_table** returns the set of storage option name/value pairs (**option\_name/option\_value**) when **pg\_class.reloptions** or **pg\_attribute.attoptions** is passed.

- `pg_tablespace_databases(tablespace_oid)`

Description: Obtains the set of database OIDs that have objects in the specified tablespace.

Return type: setof oid

Note: **pg\_tablespace\_databases** allows a tablespace to be checked. It returns the set of OIDs of databases that have objects stored in the tablespace. If this function returns any rows of data, the tablespace is not empty and cannot be dropped. To display the specific objects in the tablespace, you need to connect to the databases identified by **pg\_tablespace\_databases** and query their **pg\_class** catalogs.

- `pg_tablespace_location(tablespace_oid)`  
Description: Obtains the path in the file system that this tablespace is located in.  
Return type: text
- `pg_typeof(any)`  
Description: Obtains the data type of any value.  
Return type: regtype

Note: **pg\_typeof** returns the OID of the data type of the value that is passed to it. This can be helpful for troubleshooting or dynamically constructing SQL queries. It is declared that the function returns **regtype**, which is an OID alias type (see [Object Identifier Types](#)). This means that it is the same as an OID for comparison purposes but displays as a type name.

Example:

```
gaussdb=# SELECT pg_typeof(33);
pg_typeof

integer
(1 row)

gaussdb=# SELECT typlen FROM pg_type WHERE oid = pg_typeof(33);
typlen

4
(1 row)
```

 **NOTE**

To prevent memory bloat, it is recommended that the number of nested layers of the function be less than or equal to five.

- `collation for (any)`  
Description: Obtains the collation of the parameter.  
Return type: text

Note: The expression **collation for** returns the collation of the value that is passed to it. Example:

```
gaussdb=# SELECT collation for (description) FROM pg_description LIMIT 1;
pg_collation_for

"default"
(1 row)
```

The value might be quoted and schema-qualified. If no collation is derived for the parameter expression, then a null value is returned. If the argument is not of a collatable data type, then an error is raised.

- `getdistributekey(table_name)`  
Description: Gets a distribution key for a hash table.  
Return type: text

Example:

```
gaussdb=# SELECT getdistributekey('item');
getdistributekey

i_item_sk
(1 row)
```

- `pg_extension_update_paths(name)`

Description: Returns the version update path of the specified extension. This function can be called only by system administrators.

Return type: text (source text), text (target text), text (path text)

- `pg_get_serial_sequence(tablename, colname)`

Description: Obtains the sequence of the corresponding table name and column name.

Return type: text

Example:

```
gaussdb=# SELECT * FROM pg_get_serial_sequence('t1', 'c1');
pg_get_serial_sequence

public.serial
(1 row)
```

- `pg_sequence_parameters(sequence_oid)`

Description: Obtains the parameters of a specified sequence, including the start value, minimum value, maximum value, and incremental value.

Return type: int16, int16, int16, int16, Boolean

Example:

```
gaussdb=# SELECT * FROM pg_sequence_parameters(16420);
start_value | minimum_value | maximum_value | increment | cycle_option
-----+-----+-----+-----+-----
101 | 1 | 9223372036854775807 | 1 | f
(1 row)
```

- `gs_get_sequence_parameters(sequence_oid)`

Description: Obtains the parameters of a specified sequence, including the start value, minimum value, maximum value, and incremental value. This function returns **NULL** for an unauthorized sequence.

Return type: int16, int16, int16, int16, Boolean

Example:

```
gaussdb=# SELECT * FROM gs_get_sequence_parameters(16420);
start_value | minimum_value | maximum_value | increment | cycle_option
-----+-----+-----+-----+-----
101 | 1 | 9223372036854775807 | 1 | f
(1 row)
```

- `pgxc_get_variable_info( )`

Description: Obtains variable values on the node, including **nodeName**, **nextOid**, **nextXid**, **oldestXid**, **xidVacLimit**, **oldestXidDB**, **lastExtendCSNLogpage**, **startExtendCSNLogpage**, **nextCommitSeqNo**, **latestCompleteXid**, and **startupMaxXid**.

Return type: set of `pg_variable_info`

Example:

```
gaussdb=# SELECT pgxc_get_variable_info();
pgxc_get_variable_info

(dn_6004_6005_6006,25617,141396349,2073,20000002073,15808,138111,0,127154152,141396348,104433004)
(1 row)
```

- `gs_get_index_status(schema_name, index_name)`

Description: Obtains the index status information on all nodes, including whether an index can be inserted and whether an index is available. This function is mainly used to check the index status during online index creation or when the index creation fails. The return values include **node\_name**,

**indisready**, and **indisvalid**. Only when **indisready** and **indisvalid** of indexes on all nodes are set to **true** and the index state is not changed to unusable, the current index is available.

Return type: text, Boolean, Boolean

Example:

```
gaussdb=# SELECT * FROM gs_get_index_status('public', 'index1');
 node_name | indisready | indisvalid
-----+-----+-----
 datanode1 | t | t
 datanode2 | t | t
 coordinator1 | t | t
(3 row)
```

## Comment Information Functions

- `col_description(table_oid, column_number)`

Description: Obtains the comment for a table column.

Return type: text

Note: **col\_description** returns the comment for a table column, which is specified by the OID of its table and its column number.

- `obj_description(object_oid, catalog_name)`

Description: Obtains the comment for a database object.

Return type: text

Note: The two-parameter form of **obj\_description** returns the comment for a database object specified by its OID and the name of the system catalog to which it belongs. For example, **obj\_description(123456,'pg\_class')** would retrieve the comment for the table with OID 123456. The one-parameter form of **obj\_description** requires only the OID.

**obj\_description** cannot be used for table columns since columns do not have OIDs of their own.

- `obj_description(object_oid)`

Description: Obtains the comment for a database object.

Return type: text

- `shobj_description(object_oid, catalog_name)`

Description: Obtains the comment for a shared database object.

Return type: text

Note: **shobj\_description** is used just like **obj\_description**, except that the former is used for shared objects. Some system catalogs are global to all databases in the cluster, and the comments for objects in them are stored globally as well.

## XIDs and Snapshots

Internal XIDs are 64 bits. **txid\_snapshot**, data type used by these functions, stores information about XID visibility at a particular moment in time. [Table 7-54](#) describes its components.

**Table 7-54** Snapshot components

| Name     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| xmin     | Earliest XID ( <b>txid</b> ) that is still active. All earlier transactions will either be committed and visible, or rolled back.                                                                                                                                                                                                                                                                                                                                                                         |
| xmax     | First as-yet-unassigned <b>txid</b> . All txids greater than or equal to this are not yet started as of the time of the snapshot, so they are invisible.                                                                                                                                                                                                                                                                                                                                                  |
| xip_list | Active <b>txids</b> at the time of the snapshot. The list includes only those active <b>txids</b> between <b>xmin</b> and <b>xmax</b> ; there might be active <b>txids</b> higher than <b>xmax</b> . A <b>txid</b> that is greater than or equal to <b>xmin</b> and less than <b>xmax</b> and that is not in this list was already completed at the time of the snapshot, and is either visible or rolled back according to its commit status. The list does not include <b>txids</b> of subtransactions. |

The textual representation of **txid\_snapshot** is **xmin:xmax:xip\_list**.

For example, **10:20:10,14,15** means **xmin=10**, **xmax=20**, **xip\_list=10, 14, 15**.

The following functions provide server transaction information in an exportable form. These functions are mainly used to determine which transactions were committed between two snapshots.

- `pgxc_is_committed(transaction_id)`  
Description: Specifies whether the given XID is committed or ignored. **NULL** indicates the unknown status (it can be running, ready, frozen, or other status).  
Return type: Boolean
- `txid_current()`  
Description: Obtains the current XID.  
Return type: bigint
- `gs_txid_oldestxmin()`  
Description: Obtains the minimum XID (specified by **oldesxmin**).  
Return type: bigint
- `txid_current_snapshot()`  
Description: Obtains the current snapshot.  
Return type: txid\_snapshot
- `txid_snapshot_xip(txid_snapshot)`  
Description: Obtains in-progress XIDs in a snapshot.  
Return type: setof bigint
- `txid_snapshot_xmax(txid_snapshot)`  
Description: Obtains **xmax** of snapshots.  
Return type: bigint
- `txid_snapshot_xmin(txid_snapshot)`  
Description: Obtains **xmin** of snapshots.

- Return type: bigint
- `txid_visible_in_snapshot(bigint, txid_snapshot)`

Description: Specifies whether the XID is visible in a snapshot (do not use subtransaction IDs).

Return type: Boolean
  - `get_local_prepared_xact()`

Description: Obtains the two-phase residual transaction information of the current node, including the XID, GID of the two-phase transaction, prepared time, owner OID, database OID, and node name of the current node.

Return type: xid, text, timestamptz, oid, oid, text
  - `get_remote_prepared_xacts()`

Description: Obtains the two-phase residual transaction information of all remote nodes, including the XID, GID of the two-phase transaction, prepared time, owner name, database name, and node name.

Return type: xid, text, timestamptz, name, name, text
  - `global_clean_prepared_xacts(text, text)`

Description: Concurrently cleans two-phase residual transactions. Only the `gs_clean` tool can call this function for cleaning. In other situations, **false** is returned.

Return type: Boolean
  - `pgxc_stat_get_wal_senders()`

Description: Returns the sent logs of all primary DN's and the received logs of their corresponding standby DN's in the cluster. Only users with the **system admin** or **monitor admin** permission can use this function.

The output is as follows:

**Table 7-55** pgxc\_stat\_get\_wal\_senders parameter description

| Column                | Description                                     |
|-----------------------|-------------------------------------------------|
| nodename              | Instance name                                   |
| sender_pid            | PID of the thread for sending logs              |
| local_role            | Instance role                                   |
| peer_role             | Role of the instance on the receiver            |
| peer_state            | Status of the instance on the receiver          |
| state                 | Synchronization status between instances        |
| sender_sent_location  | Location where the sender sends logs            |
| sender_write_location | Location where the sender writes logs           |
| sender_flush_location | Location where the sender flushes logs to disks |

| Column                     | Description                                                                                                                                                                                              |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sender_replay_location     | Log location of the current instance. If the DN is a primary DN, the location is the value of <b>sender_flush_location</b> . Otherwise, the location where the current instance log is replayed is used. |
| receiver_received_location | Location where the receiver receives logs                                                                                                                                                                |
| receiver_write_location    | Location where the receiver writes logs                                                                                                                                                                  |
| receiver_flush_location    | Location where the receiver flushes logs to disks                                                                                                                                                        |
| receiver_replay_location   | Location where the receiver replays logs                                                                                                                                                                 |

- pgxc\_stat\_get\_wal\_senders\_status()  
Description: Returns the receiving status of transaction logs on all nodes. Only users with the **system admin** or **monitor admin** permission can use this function.

The return values are as follows:

**Table 7-56** pgxc\_stat\_get\_wal\_senders\_status parameter description

| Column                     | Description                            |
|----------------------------|----------------------------------------|
| nodename                   | Name of the primary node               |
| source_ip                  | IP address of the primary node         |
| source_port                | Port of the primary node               |
| dest_ip                    | IP address of the standby node         |
| dest_port                  | Port of the standby node               |
| sender_pid                 | PID of the sending thread              |
| local_role                 | Type of the primary node               |
| peer_role                  | Type of the standby node               |
| peer_state                 | Status of the standby node             |
| state                      | WAL sender status                      |
| sender_sent_location       | Sending position of the primary node   |
| sender_write_location      | Writing position of the primary node   |
| sender_replay_location     | Redo position of the primary node      |
| receiver_received_location | Receiving position of the standby node |

| Column                   | Description                           |
|--------------------------|---------------------------------------|
| receiver_write_location  | Writing position of the standby node  |
| receiver_flush_location  | Flushing location of the standby node |
| receiver_replay_location | Redo location of the standby node     |

- `gs_get_next_xid_csn()`  
Description: Returns the values of **next\_xid** and **next\_csn** on all nodes globally.  
The return values are as follows:

**Table 7-57** `gs_get_next_xid_csn` parameter description

| Column   | Description                                     |
|----------|-------------------------------------------------|
| nodename | Node name                                       |
| next_xid | ID of the next transaction on the current node. |
| next_csn | Next CSN of the current node.                   |

- `pg_control_system()`  
Description: Returns the status of the system control file.  
Return type: SETOF record
- `pg_control_checkpoint()`  
Description: Returns the system checkpoint status.  
Return type: SETOF record
- `get_prepared_pending_xid`  
Description: Returns **nextxid** when restoration is complete.  
Parameter: nan  
Return type: text
- `pg_clean_region_info`  
Description: Clears the region map.  
Parameter: nan  
Return type: character varying
- `pg_get_replication_slot_name`  
Description: Obtains the slot name.  
Parameter: nan  
Return type: text
- `pg_get_running_xacts`  
Description: Obtains running **xact**.  
Parameter: nan

Return type: handle integer, gxid xid, state tinyint, node text, xmin xid, vacuum Boolean, timeline bigint, prepare\_xid xid, pid bigint, next\_xid xid

**Table 7-58** pg\_get\_running\_xacts return parameters

| Column      | Description                                                                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle      | Handle corresponding to the transaction in GTM.                                                                                                                      |
| gxid        | Transaction ID.                                                                                                                                                      |
| state       | Transaction status. <ul style="list-style-type: none"> <li>• <b>3</b>: prepared.</li> <li>• <b>0</b>: starting.</li> </ul>                                           |
| node        | Node name.                                                                                                                                                           |
| xmin        | Minimum transaction ID on the node.                                                                                                                                  |
| vacuum      | Specifies whether the current transaction is lazy vacuum. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul> |
| timeline    | Number of database restarts.                                                                                                                                         |
| prepare_xid | ID of the transaction in the <b>prepared</b> state (the value is <b>0</b> if the state is not <b>prepared</b> ).                                                     |
| pid         | Thread ID corresponding to the transaction.                                                                                                                          |
| next_xid    | Minimum CSN of a local active transaction.                                                                                                                           |

- **pg\_get\_variable\_info**  
Description: Obtains the shared memory variable cache.  
Parameter: nan  
Return type: node\_name text, nextOid oid, nextXid xid, oldestXid xid, xidVacLimit xid, oldestXidDB oid, lastExtendCSNLogpage xid, startExtendCSNLogpage xid, nextCommitSeqNo xid, latestCompletedXid xid, and startupMaxXid xid
- **pg\_get\_xidlimit**  
Description: Obtains XID information from the shared memory.  
Parameter: nan  
Return type: nextXid xid, oldestXid xid, xidVacLimit xid, xidWarnLimit xid, xidStopLimit xid, xidWrapLimit xid, oldestXidDB oid
- **pg\_relation\_compression\_ratio**  
Description: Queries the compression rate of a table. By default, **1.0** is returned.  
Parameter: text  
Return type: real
- **pg\_relation\_with\_compression**

- Description: Specifies whether a table is compressed.  
Parameter: text  
Return type: Boolean
- `pg_stat_file_recursive`  
Description: Lists all files in a path.  
Parameter: location text  
Return type: path text, filename text, size bigint, isdir Boolean
  - `pg_stat_get_activity_for temptable`  
Description: Returns records of backend processes related to the temporary table.  
Parameter: nan  
Return type: datid oid, timelineid integer, tempid integer, and sessionid bigint
  - `pg_stat_get_activity_ng`  
Description: Returns records of backend processes related to nodegroup.  
Parameter: pid bigint  
Return type: datid oid, pid bigint, sessionid bigint, and node\_group text
  - `pg_stat_get_cgroup_info`  
Description: Returns Cgroup information.  
Parameter: nan  
Return type: cgroup\_name text, percent integer, usage\_percent integer, shares bigint, usage bigint, cpuset text, relpath text, valid text, node\_group text
  - `pg_stat_get_realtime_info_internal`  
Description: Returns real-time information. Currently, this API is unavailable. **FailedToGetSessionInfo** is returned.  
Parameter: oid, oid, bigint, cstring, oid  
Return type: text
  - `pg_stat_get_session_wlmstat`  
Description: Returns the load information of the current session.  
Parameter: pid integer  
Return type: datid oid, threadid bigint, sessionid bigint, threadpid integer, usesysid oid, appname text, query text, priority bigint, block\_time bigint, elapsed\_time bigint, total\_cpu\_time bigint, skew\_percent integer, statement\_mem integer, active\_points integer, dop\_value integer, current\_cgroup text, current\_status text, enqueue\_state text, attribute text, is\_plana Boolean, node\_group text, srespool name
  - `pg_stat_get_wlm_instance_info`  
Description: Returns the load information of the current instance.  
Parameter: nan  
Return type: instancename text, timestamp, timestamp with time zone, used\_cpu integer, free\_memory integer, used\_memory integer, io\_await double precision, io\_util double precision, disk\_read double precision, disk\_write double precision, process\_read bigint, process\_write bigint, logical\_read bigint, logical\_write bigint, read\_counts bigint, write\_counts bigint

- `pg_stat_get_wlm_instance_info_with_cleanup`  
Description: Returns the load information of the current instance and saves the information to the system catalog.  
Parameter: nan  
Return type: instancename text, timestamp, timestamp with time zone, used\_cpu integer, free\_memory integer, used\_memory integer, io\_await double precision, io\_util double precision, disk\_read double precision, disk\_write double precision, process\_read bigint, process\_write bigint, logical\_read bigint, logical\_write bigint, read\_counts bigint, write\_counts bigint
- `pg_stat_get_wlm_node_resource_info`  
Description: Obtains the resource information of the current node.  
Parameter: nan  
Return type: min\_mem\_util integer, max\_mem\_util integer, min\_cpu\_util integer, max\_cpu\_util integer, min\_io\_util integer, max\_io\_util integer, used\_mem\_rate integer
- `pg_stat_get_wlm_operator_info`  
Description: Obtains the operator information of the execution plan from the internal hash table.  
Parameter: nan  
Return type: queryid bigint, pid bigint, plan\_node\_id integer, plan\_node\_name text, start\_time timestamp with time zone, duration bigint, query\_dop integer, estimated\_rows bigint, tuple\_processed bigint, min\_peak\_memory integer, max\_peak\_memory integer, average\_peak\_memory integer, memory\_skew\_percent integer, min\_spill\_size integer, max\_spill\_size integer, average\_spill\_size integer, spill\_skew\_percent integer, min\_cpu\_time bigint, max\_cpu\_time bigint, total\_cpu\_time bigint, cpu\_skew\_percent integer, warning text
- `pg_stat_get_wlm_realtime_operator_info`  
Description: Obtains the operator information of the real-time execution plan from the internal hash table.  
Parameter: nan  
Return type: queryid bigint, pid bigint, plan\_node\_id integer, plan\_node\_name text, start\_time timestamp with time zone, duration bigint, status text, query\_dop integer, estimated\_rows bigint, tuple\_processed bigint, min\_peak\_memory integer, max\_peak\_memory integer, average\_peak\_memory integer, memory\_skew\_percent integer, min\_spill\_size integer, max\_spill\_size integer, average\_spill\_size integer, spill\_skew\_percent integer, min\_cpu\_time bigint, max\_cpu\_time bigint, total\_cpu\_time bigint, cpu\_skew\_percent integer, warning text
- `pg_stat_get_wlm_realtime_session_info`  
Description: Returns the load information of the real-time session.  
Parameter: nan  
Return type: nodename text, threadid bigint, block\_time bigint, duration bigint, estimate\_total\_time bigint, estimate\_left\_time bigint, schemaname text, query\_band text, spill\_info text, control\_group text, estimate\_memory integer, min\_peak\_memory integer, max\_peak\_memory integer, average\_peak\_memory integer, memory\_skew\_percent integer, min\_spill\_size

integer, max\_spill\_size integer, average\_spill\_size integer, spill\_skew\_percent integer, min\_dn\_time bigint, max\_dn\_time bigint, average\_dn\_time bigint, dntime\_skew\_percent integer, min\_cpu\_time bigint, max\_cpu\_time bigint, total\_cpu\_time bigint, cpu\_skew\_percent integer, min\_peak\_iops integer, max\_peak\_iops integer, average\_peak\_iops integer, iops\_skew\_percent integer, warning text, query text, query\_plan text, cpu\_top1\_node\_name text, cpu\_top2\_node\_name text, cpu\_top3\_node\_name text, cpu\_top4\_node\_name text, cpu\_top5\_node\_name text, mem\_top1\_node\_name text, mem\_top2\_node\_name text, mem\_top3\_node\_name text, mem\_top4\_node\_name text, mem\_top5\_node\_name text, cpu\_top1\_value bigint, cpu\_top2\_value bigint, cpu\_top3\_value bigint, cpu\_top4\_value bigint, cpu\_top5\_value bigint, mem\_top1\_value bigint, mem\_top2\_value bigint, mem\_top3\_value bigint, mem\_top4\_value bigint, mem\_top5\_value bigint, top\_mem\_dn text, top\_cpu\_dn text

- `pg_stat_get_wlm_session_info_internal`  
Description: Returns the session load information.  
Parameter: oid, bigint, bigint, oid  
Return type: SETOF text
- `pg_stat_get_wlm_session_iostat_info`  
Description: Returns the session load I/O information.  
Parameter: nan  
Return type: threadid bigint, maxcurr\_iops integer, mincurr\_iops integer, maxpeak\_iops integer, minpeak\_iops integer, iops\_limits integer, io\_priority integer, and curr\_io\_limits integer
- `pg_stat_get_wlm_statistics`  
Description: Returns session load statistics.  
Parameter: nan  
Return type: statement text, block\_time bigint, elapsed\_time bigint, total\_cpu\_time bigint, qualification\_time bigint, skew\_percent integer, control\_group text, status text, action text
- `pg_test_err_contain_err`  
Description: Tests the error type and return information.  
Parameter: integer  
Return type: void
- `pv_session_memory_detail_tp`  
Description: Returns the memory usage of the session. For details, see **pv\_session\_memory\_detail**.  
Parameter: nan  
Return type: sessid text, sesstype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint
- `gs_get_table_distribution`  
Description: Returns the distribution of table data on each DN.  
Parameter: table\_name text, schema\_name text  
Return type: text
- `pv_builtin_functions`

Description: Displays information about all built-in system functions.

Parameter: nan

Return type: proname name, pronamespace oid, proowner oid, prolang oid, procost real, prorows real, provariadic oid, protransform regproc, proisagg Boolean, proiswindow Boolean, prosecdef Boolean, proleakproof Boolean, proisstrict Boolean, proretset Boolean, provolatile "char", pronargs smallint, pronargdefaults smallint, prorettytype oid, proargtypes oidvector, proallargtypes integer[], proargmodes "char"[], proargnames text[], proargdefaults pg\_node\_tree, prosrc text, probin text, proconfig text[], proacl aclitem[], prodefaultargpos int2vector, fencedmode Boolean, proshippable Boolean, propackage Boolean, oid oid

- `pv_thread_memory_detail`

Description: Returns the memory information of each thread.

Parameter: nan

Return type: threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- `pg_shared_memory_detail`

Description: Returns usage information about all generated shared memory contexts. For details about each column, see [SHARED\\_MEMORY\\_DETAIL](#).

Parameter: nan

Return type: contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- `pgxc_get_running_xacts`

Description: Returns information about running transactions on each node in the cluster. The field content is the same as that in [PGXC\\_RUNNING\\_XACTS](#). Only users with the system admin or monitor admin permission can view the information.

Parameter: nan

Return type: SETOF record

- `pgxc_snapshot_status()`

Description: Returns the status of key memory information in the GTM in GTM mode for fault locating. GTM-Free and GTM-Lite do not support this function.

Parameter: nan

Return type: xmin xid, xmax xid, xcnt int, oldestxmin xid, next\_xid xid, timeline int, active\_thread\_num int, max\_active\_thread\_num int, snapshot\_num int, snapshot\_totalsize bigint

The return values are described as follows:

**Table 7-59** `get_gtm_lite_status` return parameters

| Column | Description                    |
|--------|--------------------------------|
| xmin   | Minimum active XID in the GTM. |

| Column                | Description                                                                                                         |
|-----------------------|---------------------------------------------------------------------------------------------------------------------|
| xmax                  | Largest XID committed in the GTM plus 1. Transactions whose IDs are greater than or equal to this value are active. |
| xcnt                  | Number of active transactions in the GTM.                                                                           |
| oldestxmin            | ID of the earliest accessed transaction in the GTM.                                                                 |
| next_xid              | ID of the next transaction allocated by the GTM.                                                                    |
| timeline              | Current time line in the GTM.                                                                                       |
| active_thread_num     | Number of active worker threads in the GTM.                                                                         |
| max_active_thread_num | Peak number of worker threads in the GTM within one minute.                                                         |
| snapshot_num          | Number of snapshots delivered by the GTM within one minute.                                                         |
| snapshot_totalsize    | Total size of snapshots delivered by the GTM within one minute.                                                     |

- `get_gtm_lite_status()`  
Description: Returns the backup XID and CSN in the GTM for fault locating. This system function is not supported in GTM-Free mode.  
The return values are as follows:

**Table 7-60** `get_gtm_lite_status` return parameters

| Column     | Description                   |
|------------|-------------------------------|
| backup_xid | Backup XID in the GTM.        |
| csn        | Latest CSN issued by the GTM. |

- `adm_hist_snapshot_func()`  
Description: Returns information about the snapshot execution time. To access this function, set the **enable\_wdr\_snapshot** parameter to **on** and obtain the permission to access the **snapshot schema**, **snapshot table**, and **tables\_snap\_timestamp** tables.  
Parameter: nan  
Return type: `snap_id` bigint, `dbid` oid, `begin_interval_time` timestamp(3), `end_interval_time` timestamp(3), `flush_elapsed` interval day(5) to second(1), `begin_interval_time_tz` timestamp(3) with time zone, `end_interval_time_tz` timestamp(3) with time zone
- `global_sql_patch_func()`  
Description: SQL patch information on each global node, which is used to return the result of the `global_sql_patch` view.

The output is as follows:

**Table 7-61** global\_sql\_patch\_func return parameters

| Name                 | Type    | Description                                                                                                                                                                                                                                                                                                                             |
|----------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| node_name            | text    | Name of the node where the SQL patch is located.                                                                                                                                                                                                                                                                                        |
| patch_name           | name    | Patch name.                                                                                                                                                                                                                                                                                                                             |
| unique_sql_id        | bigint  | Global unique query ID.                                                                                                                                                                                                                                                                                                                 |
| owner                | oid     | ID of the user who creates the patch.                                                                                                                                                                                                                                                                                                   |
| enable               | boolean | Determines whether the patch takes effect.                                                                                                                                                                                                                                                                                              |
| status               | "char"  | Patch status (reserved field).                                                                                                                                                                                                                                                                                                          |
| abort                | boolean | Determines whether the value is <b>AbortHint</b> .                                                                                                                                                                                                                                                                                      |
| hint_string          | text    | Hint text.                                                                                                                                                                                                                                                                                                                              |
| description          | text    | Patch description.                                                                                                                                                                                                                                                                                                                      |
| parent_unique_sql_id | bigint  | Globally unique ID of the outer statement of the SQL statement for which the patch takes effect. The value of this parameter is <b>0</b> for statements outside a stored procedure. For statements inside the stored procedure, the value of this parameter is the globally unique ID of the statement that calls the stored procedure. |

- `gs_get_current_version()`  
Description: Returns the current compilation mode based on the current compilation macro. **'M'** is returned.  
Parameter: nan  
Return type: char
- `gs_get_kernel_info()`  
Description: Global transaction information on each primary CN and DN.  
The return values are as follows:

**Table 7-62** gs\_get\_kernel\_info parameter description

| Name      | Type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| node_name | text | Node name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| module    | text | Module name, including: <ul style="list-style-type: none"> <li>• <b>XACT</b> (transaction module)</li> <li>• <b>STANDBY</b> (standby module)</li> <li>• <b>UNDO</b> (undo module)</li> <li>• <b>HOTPATH</b> (hot patch module)</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| name      | text | Name of the key data in the memory state, including: <ul style="list-style-type: none"> <li>• <b>startup_max_xid</b> (maximum XID when a process is started)</li> <li>• <b>recent_local_xmin</b> (minimum XID of a local active transaction)</li> <li>• <b>recent_global_xmin</b> (minimum XID of a global active transaction)</li> <li>• <b>standby_xmin</b> (minimum XID of an active transaction on the standby node)</li> <li>• <b>standby_redo_cleanup_xmin</b> (minimum XID of cleanup logs during redo on the standby node)</li> <li>• <b>standby_redo_cleanup_xmin_lsn</b> (LSN of the minimum XID of cleanup logs during redo on the standby node)</li> <li>• <b>local_csn_min</b> (minimum CSN of a local active transaction)</li> <li>• <b>replication_slot_xmin</b> (minimum XID of a replication slot)</li> <li>• <b>replication_slot_catalog_xmin</b> (minimum XID of a catalog replication slot)</li> <li>• <b>global_recycle_xid</b> (minimum XID of a global undo recycling transaction)</li> <li>• <b>global_frozen_xid</b> (minimum XID of a globally frozen transaction)</li> <li>• <b>hotpatch_additional_info</b> (reserved column for hot patches)</li> </ul> |
| value     | text | Value of the key data in the memory state.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

Example:

```
gaussdb=# select * from gs_get_kernel_info();
node_name | module | name | value
-----+-----+-----+-----
```

```

coordinator1 | XACT | startup_max_xid | 16488
coordinator1 | XACT | recent_local_xmin | 16504
coordinator1 | XACT | recent_global_xmin | 16503
coordinator1 | STANDBY | standby_xmin | 0
coordinator1 | STANDBY | standby_redo_cleanup_xmin | 0
coordinator1 | STANDBY | standby_redo_cleanup_xmin_lsn | 0/0
coordinator1 | XACT | local_csn_min | 6014225
coordinator1 | XACT | replication_slot_xmin | 0
coordinator1 | XACT | replication_slot_catalog_xmin | 0
coordinator1 | UNDO | global_recycle_xid | 16501
coordinator1 | XACT | global_frozen_xid | 0
coordinator1 | HOTPATH | hotpatch_additional_info |
datanode1 | XACT | startup_max_xid | 16488
datanode1 | XACT | recent_local_xmin | 15805
datanode1 | XACT | recent_global_xmin | 15805
datanode1 | STANDBY | standby_xmin | 0
datanode1 | STANDBY | standby_redo_cleanup_xmin | 0
datanode1 | STANDBY | standby_redo_cleanup_xmin_lsn | 0/0
datanode1 | XACT | local_csn_min | 6014225
datanode1 | XACT | replication_slot_xmin | 0
datanode1 | XACT | replication_slot_catalog_xmin | 0
datanode1 | UNDO | global_recycle_xid | 15805
datanode1 | XACT | global_frozen_xid | 0
datanode1 | HOTPATH | hotpatch_additional_info |
(24 row)

```

## 7.5.25 System Administration Functions

### 7.5.25.1 Configuration Settings Functions

Configuration setting functions are used for querying and modifying configuration parameters during running.

- `current_setting(setting_name)`

Description: Specifies the current setting.

Return type: text

Note: **current\_setting** obtains the current setting of **setting\_name** by query. It is equivalent to the **SHOW** statement. For example:

```
gaussdb=# SELECT current_setting('datestyle');
```

```

current_setting

ISO, MDY
(1 row)

```

- `set_working_grand_version_num_manually(tmp_version)`

Description: Upgrades new features of GaussDB by switching the authorization version.

Return type: void

- `shell_in(type)`

Description: Inputs a route for the shell type that has not yet been filled.

Return type: void

- `shell_out(type)`

Description: Outputs a route for the shell type that has not yet been filled.

Return type: void

- `set_config(setting_name, new_value, is_local)`

Description: Sets the parameter and returns a new value.

Return type: text

Note: **set\_config** sets the parameter **setting\_name** to **new\_value**. If **is\_local** is **true**, **new\_value** will only apply to the current transaction. If you want **new\_value** to apply to the current session, use **false** instead. This function is equivalent to the SQL statement SET.

Example:

```
gaussdb=# SELECT set_config('log_statement_stats', 'off', false);

 set_config

off
(1 row)
```

## 7.5.25.2 Universal File Access Functions

Universal file access functions provide local access APIs for files on a database server. Only files in the database cluster directory and the **log\_directory** directory can be accessed. A relative path is used for files in the cluster directory, and a path matching the **log\_directory** configuration is used for log files. Only database initialization users can use these functions.

- **pg\_ls\_dir**(dirname text)

Description: Lists files in a directory.

Return type: setof text

Note: **pg\_ls\_dir** returns all the names in the specified directory, except the special entries "." and "..".

Example:

```
gaussdb=# SELECT pg_ls_dir('./');
 pg_ls_dir

.postgresql.conf.swp
postgresql.conf
pg_tblspc
PG_VERSION
pg_ident.conf
core
server.crt
pg_serial
pg_twophase
postgresql.conf.lock
pg_stat_tmp
pg_notify
pg_subtrans
pg_ctl.lock
pg_xlog
pg_clog
base
pg_snapshots
postmaster.opts
postmaster.pid
server.key.rand
server.key.cipher
pg_multixact
pg_errorinfo
server.key
pg_hba.conf
pg_replslot
.pg_hba.conf.swp
cacert.pem
```

```
pg_hba.conf.lock
global
gaussdb.state
(32 rows)
```

- `pg_read_file(filename text, offset bigint, length bigint)`

Description: Returns the content of a text file.

Return type: text

Note: `pg_read_file` returns part of a text file. It can return a maximum of *length* bytes from *offset*. The actual size of fetched data is less than *length* if the end of the file is reached first. If *offset* is negative, it is the length rolled back from the file end. If *offset* and *length* are omitted, the entire file is returned.

Example:

```
gaussdb=# SELECT pg_read_file('postmaster.pid',0,100);
 pg_read_file
```

```

53078 +
/srv/BigData/testdir/data1/coordinator+
1500022474 +
8000 +
/var/run/FusionInsight +
localhost +
2
(1 row)
```

- `pg_read_binary_file(filename text [, offset bigint, length bigint,missing_ok boolean])`

Description: Returns the content of a binary file.

Return type: bytea

Note: `pg_read_binary_file` is similar to `pg_read_file`, except that the result is a **bytea** value; accordingly, no encoding checks are performed. In combination with the `convert_from` function, this function can be used to read a file in a specified encoding.

```
gaussdb=# SELECT convert_from(pg_read_binary_file('filename'), 'UTF8');
```

- `pg_stat_file(filename text)`

Description: Returns status information about a file.

Return type: record

Note: `pg_stat_file` returns a record containing the file size, last access timestamp, last modification timestamp, last file status change timestamp, and a **Boolean** value indicating if it is a directory. Typical usages are as follows:

```
gaussdb=# SELECT * FROM pg_stat_file('filename');
gaussdb=# SELECT (pg_stat_file('filename')).modification;
```

Example:

```
gaussdb=# SELECT convert_from(pg_read_binary_file('postmaster.pid'), 'UTF8');
 convert_from
```

```

4881 +
/srv/BigData/gaussdb/data1/coordinator+
1496308688 +
25108 +
/opt/huawei/Bigdata/gaussdb/gaussdb_tmp +
* +
25108001 43352069 +
(1 row)
```

```
gaussdb=# SELECT * FROM pg_stat_file('postmaster.pid');
 size | access | modification | change
 | creation | isdir
-----+-----+-----+-----
+-----+-----+-----+-----
 117 | 2017-06-05 11:06:34+08 | 2017-06-01 17:18:08+08 | 2017-06-01 17:18:08+08
 | | f
(1 row)
gaussdb=# SELECT (pg_stat_file('postmaster.pid')).modification;
 modification

2017-06-01 17:18:08+08
(1 row)
```

### 7.5.25.3 Server Signal Functions

Server signal functions send control signals to other server processes. Only a system administrator has the permission to execute the following functions:

- `pg_cancel_backend(pid int)`  
Description: Cancels a statement that is being executed by a backend thread.  
Return type: Boolean  
Note: **pg\_cancel\_backend** sends a query cancellation (SIGINT) signal to the backend process identified by **pid**. The PID of an active backend process can be found in the **pid** column of the **pg\_stat\_activity** view, or can be found by listing the database process using **ps** on the server. A user with the SYSADMIN permission, the owner of the database connected to the backend process, the owner of the backend process, or a user who inherits permissions of the built-in role `gs_role_signal_backend` has the permission to use this function.
- `pg_cancel_session(pid bigint, sessionid bigint)`  
Description: Cancels the statement that is being executed by an active session in thread pool mode.  
Return type: Boolean  
Note: The input parameters of `pg_cancel_session` can be queried using the **pid** and **sessionid** columns in `pg_stat_activity`. The input parameters can be used to clear the statements that are being executed by active sessions in thread pool mode. When the input parameters **pid** and **sessionid** are the same and both are thread IDs, the function of `pg_cancel_backend` is the same as that of `pg_cancel_backend`.
- `pg_cancel_invalid_query()`  
Description: Cancels the invalid query of a backend.  
Return type: Boolean  
Note: Only system administrators have the permission to cancel queries that are running in the backend of a degraded GTM.
- `pg_reload_conf()`  
Description: Causes all server processes to reload their configuration files.  
Return type: Boolean  
Note: **pg\_reload\_conf** sends a SIGHUP signal to the server. As a result, all server processes reload their configuration files.
- `pg_rotate_logfile()`  
Description: Rotates the log files of the server.

Return type: Boolean

Note: `pg_rotate_logfile` sends a signal to the log file manager, instructing the manager to immediately switch to a new output file. This function works only when **`redirect_stderr`** is used for log output. Otherwise, no log file manager subprocess is generated.

- `pg_terminate_session(pid bigint, sessionid bigint)`

Description: Terminates a backend session in thread pool mode.

Return type: Boolean

Note: The input parameters of this function can be queried using the **`pid`** and **`sessionid`** fields in **`pg_stat_activity`**. A user with the SYSADMIN permission, the owner of the database connected to the session, the owner of the session, or a user who inherits permissions of the built-in role `gs_role_signal_backend` has the permission to use this function. If the target session fails to be terminated after this function is executed, the socket connection between the session and the client is forcibly closed.

---

#### NOTICE

When the input parameters **`pid`** and **`sessionid`** are the same and both are thread IDs, this function can terminate non-thread pool threads and active thread pool threads.

When the input parameters **`pid`** and **`sessionid`** are different, this function can terminate active sessions or close inactive sessions and the socket connection of the client.

- 
- `pg_terminate_active_session_socket(pid int64, sessionid int64)`

Description: Closes the socket connection between an active session and the client.

Return type: Boolean

Note: Each of these functions returns **`true`** if they are successful and **`false`** otherwise. A user with the SYSADMIN permission, the owner of the database connected to the backend thread, the owner of the backend thread, or a user who inherits permissions of the built-in role `gs_role_signal_backend` can use this function.

- `pg_terminate_backend(pid int)`

Description: Terminates a backend thread.

Return type: Boolean

Note: Each of these functions returns **`true`** if they are successful and **`false`** otherwise. A user with the SYSADMIN permission, the owner of the database connected to the backend thread, the owner of the backend thread, or a user who inherits permissions of the built-in role `gs_role_signal_backend` can use this function. If the target session fails to be terminated after this function is executed, the socket connection between the session and the client is forcibly closed.

**NOTICE**

This function can terminate non-thread pool threads and active thread pool threads, but cannot terminate inactive thread pool threads.

Example:

```
gaussdb=# SELECT pid from pg_stat_activity;
 pid

140657876268816
(1 rows)

gaussdb=# SELECT pg_terminate_backend(140657876268816);
 pg_terminate_backend

t
(1 row)
```

## 7.5.25.4 Backup and Restoration Control Functions

### Backup Control Functions

Backup control functions help with online backup.

- `pg_create_restore_point(name text)`

Description: Creates a named point for performing a restore (requires an administrator role).

Return type: text

Note: **pg\_create\_restore\_point** creates a named transaction log record that can be used as a restoration target, and returns the corresponding transaction log location. The given name can then be used with **recovery\_target\_name** to specify the point up to which restoration will proceed. Avoid creating multiple restoration points with the same name, since restoration will stop at the first one whose name matches the restoration target.

- `pg_current_xlog_location()`

Description: Obtains the write position of the current transaction log.

Return type: text

Note: **pg\_current\_xlog\_location** displays the write position of the current transaction log in the same format as those of the previous functions. Read-only operations do not require SYSADMIN permissions.

- `pg_current_xlog_insert_location()`

Description: Obtains the insert position of the current transaction log.

Return type: text

Note: **pg\_current\_xlog\_insert\_location** displays the insert position of the current transaction log. The insertion point is the logical end of the transaction log at any instant, while the write location is the end of what has been written out from the server's internal buffers. The write position is the end that can be detected externally from the server. This operation can be performed to archive only some of completed transaction log files. The insert position is mainly used for commissioning the server. Read-only operations do not require SYSADMIN permissions.

- `gs_current_xlog_insert_end_location()`  
Description: Obtains the insert position of the current transaction log.  
Return type: text  
Note: **gs\_current\_xlog\_insert\_end\_location** displays the insert position of the current transaction log.
- `pg_start_backup(label text[, fast boolean ])`  
Description: Starts to perform online backup. (An administrator, replication role, or O&M administrator must enable **operation\_mode**.) Label strings starting with **gs\_roach** are reserved and can be used only by the internal backup tool GaussRoach.  
Return type: text  
Note: **pg\_start\_backup** receives a user-defined backup label (usually the name of the position where the backup dump file is stored). This function writes a backup label file to the data directory of the database cluster and then returns the starting position of backed up transaction logs in text mode.  

```
gaussdb=# SELECT pg_start_backup('label_goes_here',true);
pg_start_backup

0/3000020
(1 row)
```
- `pg_stop_backup()`  
Description: Completes online backup An administrator, replication role, or O&M administrator must enable **operation\_mode**.  
Return type: text  
Note: **pg\_stop\_backup** deletes the label file created by **pg\_start\_backup** and creates a backup history file in the transaction log archive area. The history file includes the label given to **pg\_start\_backup**, the start and end transaction log locations for the backup, and the start and end time of the backup. The return value is the backup's ending transaction log location. After the end position is calculated, the insert position of the current transaction log automatically goes ahead to the next transaction log file. The ended transaction log file can be immediately archived so that backup is complete.
- `pg_switch_xlog()`  
Description: Switches to a new transaction log file An administrator or O&M administrator must enable **operation\_mode**.  
Return type: text  
Note: **pg\_switch\_xlog** moves to the next transaction log file so that the current log file can be archived (if continuous archive is used). The return value is the ending transaction log location + 1 within the just-completed transaction log file. If there has been no transaction log activity since the last transaction log switchover, `pg_switch_xlog` does not move but returns the start location of the transaction log file currently in use.
- `pg_xlogfile_name(location text)`  
Description: Converts the position string in a transaction log to a file name.  
Return type: text  
Note: **pg\_xlogfile\_name** extracts only the transaction log file name. If the given transaction log position is the transaction log file border, a transaction log file name will be returned for both the two functions. This is usually the

desired behavior for managing transaction log archiving, since the preceding file is the last one that currently needs to be archived.

- `pg_xlogfile_name_offset(location text)`

Description: Converts the position string in a transaction log to a file name and returns the byte offset in the file.

Return type: text, integer

Note: **pg\_xlogfile\_name\_offset** can extract transaction log file names and byte offsets from the returned results of the preceding functions. Example:

```
gaussdb=# SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());
NOTICE: pg_stop_backup cleanup done, waiting for required WAL segments to be archived
NOTICE: pg_stop_backup complete, all required WAL segments have been archived
 file_name | file_offset
-----+-----
000000010000000000000003 | 272
(1 row)
```

- `pg_xlog_location_diff(location text, location text)`

Description: Calculates the difference in bytes between two transaction log locations.

Return type: numeric

- `pg_cbm_start_tracked_location()`

Description: Queries the start LSN parsed by CBM.

Return type: text

- `pg_cbm_tracked_location()`

Description: Queries the LSN parsed by CBM.

Return type: text

- `pg_cbm_get_merged_file(startLSNArg text, endLSNArg text)`

Description: Combines CBM files within the specified LSN range into one and returns the name of the combined file.

Return type: text

Note: Only SYSADMIN or OPRADMIN can obtain the CBM combination file.

- `pg_cbm_get_changed_block(startLSNArg text, endLSNArg text)`

Description: Combines CBM files within the specified LSN range into a table and return records of this table.

Return type: records

Note: The table columns returned by `pg_cbm_get_changed_block` include the start LSN, end LSN, tablespace OID, database OID, table relfilenode, table fork number, whether the table is a system catalog, whether the table is deleted, whether the table is created, whether the table is truncated, number of pages in the truncated table, number of modified pages, and list of modified page numbers.

- `pg_cbm_recycle_file(targetLSNArg text)`

Description: Deletes the CBM files that are no longer used and returns the first LSN after the deletion.

Return type: text

- `pg_cbm_force_track(targetLSNArg text,timeOut int)`

Description: Forcibly executes the CBM trace to the specified Xlog position and returns the Xlog position of the actual trace end point.

- Return type: text
- `pg_enable_delay_ddl_recycle()`  
Description: Enables DDL delay and returns the Xlog position of the enabling point. An administrator or O&M administrator must enable **operation\_mode**.  
Return type: text
  - `pg_disable_delay_ddl_recycle(barrierLSNArg text, isForce bool)`  
Description: Disables DDL delay and returns the Xlog range where DDL delay takes effect. An administrator or O&M administrator must enable **operation\_mode**.  
Return type: records
  - `pg_enable_delay_xlog_recycle()`  
Description: Enables the Xlog recycling delay function for CN recovery. An administrator or O&M administrator must enable **operation\_mode**.  
Return type: void
  - `pg_disable_delay_xlog_recycle()`  
Description: Disables the Xlog recycling delay function for CN recovery. An administrator or O&M administrator must enable **operation\_mode**.  
Return type: void
  - `pg_cbm_rotate_file(rotate_lsn text)`  
Description: Forcibly switches the file after the CBM parses **rotate\_lsn**. This function is called during the build process.  
Return type: void
  - `gs_roach_stop_backup(backupid text)`  
Description: Stops a backup started by the internal backup tool GaussRoach. It is similar to the **pg\_stop\_backup system** function but is more lightweight.  
Return type: text. The content is the insertion position of the current log.
  - `gs_roach_enable_delay_ddl_recycle(backupid name)`  
Description: Enables DDL delay and returns the log position of the enabling point. It is similar to the **pg\_enable\_delay\_ddl\_recycle** system function but is more lightweight. In addition, different **backupid** values can be used to concurrently open DDL statements with delay.  
Return type: text. The content is the log location of the start point.
  - `gs_roach_disable_delay_ddl_recycle(backupid text)`  
Description: Disables DDL delay, returns the range of logs on which DDL delay takes effect. It is similar to the **pg\_enable\_delay\_ddl\_recycle** system function but is more lightweight. In addition, the DDL delay function can be disabled concurrently by specifying different backupid values.  
Return type: records. The content is the range of logs for which DDL is delayed to take effect.
  - `gs_roach_switch_xlog(request_ckpt bool)`  
Description: Switches the currently used log segment file and triggers a full checkpoint if **request\_ckpt** is set to **true**.  
Return type: text. The content is the location of the segment log.
  - `gs_block_dw_io(timeout int, identifier text)`

Description: Blocks doublewrite page flushing.

Parameters:

– timeout

Block duration.

Value range: [0,3600], in seconds. The value **0** indicates that the block duration is 0s.

– identifier

ID of the operation.

Value range: a string, supporting only uppercase letters, lowercase letters, digits, and underscores (\_).

Return type: Boolean

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation\_mode**.

- `gs_is_dw_io_blocked()`

Description: Checks whether disk flushing on the current doublewrite page is blocked. If disk flushing is blocked, **true** is returned.

Return type: Boolean

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation\_mode**.

- `gs_pitr_advance_last_updated_barrier()`

Description: In PITR mode, forcibly updates the global maximum archived recovery point uploaded to OBS/NAS last time to the current point. No input parameter is required.

Return type: text

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation\_mode**. This parameter is valid only on the first normal CN in a distributed system. The return value is the latest local maximum archived recovery point.

- `gs_pitr_clean_local_barrier_files('delete_timestamp')`

Description: Clears locally cached barrier record files.

Value range: The **delete\_timestamp** parameter is of the text type. It is a Linux timestamp and contains 10 characters.

Return type: text

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation\_mode**. The returned result is the start timestamp of the earliest barrier file on the localhost after the deletion.

- `gs_get_barrier_lsn(barrier_name text)`

Description: Obtains the LSN corresponding to the barrier created using a backup.

Return type: text

Note: Currently, only **gs\_roach\_full** and **gs\_roach\_inc** are supported. To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation\_mode**.

- `gs_gbr_relation_associated_filenode(schemaName name, tableName name)`  
Description: Returns the relfilenode of all indexes, sequences, partitions, TOASTs, and TOAST indexes related to the input table.  
Return type: records  
Note: The table columns returned by `gs_gbr_relation_associated_filenode` include the file type relkind, namespace where the file is located, relation name corresponding to the file, OID of the database where the file is located, OID of the tablespace where the file is located, and relfilenode of the file.
- `pg_create_physical_replication_slot_extern(slotname text, dummy_standby bool, extra_content text, need_recycle_xlog bool)`

---

#### NOTICE

Value range of **slotname**: a string, supporting only lowercase letters, digits, underscores (`_`), question marks (`?`), hyphens (`-`), and periods (`.`). One or two periods cannot be used alone as the replication slot name. You are advised to use an alphabetic character string as the archive slot name, and the length cannot exceed 64 characters.

---

Description: Creates an OBS or a NAS archive slot. **slotname** indicates the name of the archive slot or recovery slot. The primary and standby slots must use the same slot name. **dummy\_standby** is a reserved parameter.

**extra\_content** contains some information about the archive slot. For an OBS archive slot, the format is

**OBS;obs\_server\_ip;obs\_bucket\_name;obs\_ak;obs\_sk;archive\_path;is\_recovery;is\_vote\_replicate**, in which **OBS** indicates the archive media of the archive slot, **obs\_server\_ip** indicates the IP address of OBS, **obs\_bucket\_name** indicates the bucket name, **obs\_ak** indicates the AK of OBS, **obs\_sk** indicates the SK of OBS, **archive\_path** indicates the archive path, and **is\_recovery** specifies whether the slot is an archive slot or a recovery slot (**0**: archive slot; **1**: recovery slot). **is\_vote\_replicate** specifies whether the voting copy is archived first. The value **0** indicates that the synchronous standby node is archived first, and the value **1** indicates that the voting copy is archived first. This column is reserved in the current version and is not adapted yet. For a NAS archive slot, the format is

**NAS;archive\_path;is\_recovery;is\_vote\_replicate**. Compared with the OBS archive slot, the NAS archive slot does not have the OBS configuration information, while the meanings of other fields are the same.

If the media is not OBS or NAS, the OBS archive slot is used by default. The **extra\_content** format is

**obs\_server\_ip;obs\_bucket\_name;obs\_ak;obs\_sk;archive\_path;is\_recovery;is\_vote\_replicate**.

**need\_recycle\_xlog** specifies whether to recycle old archived logs when creating an archive slot. The value **true** indicates that old archived logs are recycled, and the value **false** indicates that old archived logs are not recycled.

Return type: records, including slotname and xlog\_position.

Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit permissions of the built-in role `gs_role_replication`. Currently, multiple archive slots cannot be created.

Examples:

Create an OBS archive slot.

```
gaussdb=# SELECT * FROM pg_create_physical_replication_slot_extern('uuid', false, 'OBS;obs.cn-north-7.ulanqab.huawei.com;dyk;19D772JBCACXX3KWS51D;*****;gaussdb_uuid/dn1;0;0', false);
slotname | xlog_position
-----+-----
 uuid |
(1 row)
```

Create a NAS archive slot.

```
gaussdb=# SELECT * FROM pg_create_physical_replication_slot_extern('uuid', false, 'NAS;/data/nas/media/gaussdb_uuid/dn1;0;0', false);
slotname | xlog_position
-----+-----
 uuid |
```

- `gs_set_obs_delete_location(delete_location text)`

Description: Sets the location where OBS archived logs can be deleted. The value of **delete\_location** is an LSN. The logs before this location have been replayed and flushed to disks and can be deleted on OBS.

Return type: `xlog_file_name text`, indicating the file name of the logs that can be deleted. The value of this parameter is returned regardless of whether OBS is deleted successfully.

```
gaussdb=# SELECT gs_set_obs_delete_location('0/54000000');
gs_set_obs_delete_location

0000000100000000000000054_00
(1 row)
```

- `gs_set_obs_delete_location_with_slotname(cstring, cstring )`

Description: Sets the location where OBS archived logs in a specified archived slot can be deleted. The first parameter indicates the LSN. The logs before this location have been replayed and flushed to disks and can be deleted on OBS. The second parameter indicates the name of the archive slot.

Return type: `xlog_file_name text`, indicating the file name of the logs that can be deleted. The value of this parameter is returned regardless of whether OBS is deleted successfully.

- `gs_get_global_barrier_status()`

Description: `gs_get_global_barrier_status` is used to query the latest global barrier archived in OBS by the primary cluster.

Return type: text

**global\_barrier\_id**: globally latest barrier ID.

**global\_achive\_barrier\_id**: globally latest archived barrier ID.

- `gs_get_global_barriers_status()`

Description: `gs_get_global_barriers_status` is used to query the latest global barrier archived in OBS by the primary cluster.

Return type: text

**slot\_name**: slot name.

**global\_barrier\_id**: globally latest barrier ID.

**global\_achive\_barrier\_id**: globally latest archived barrier ID.

- `gs_set_archive_standby_name_on_primary(text)`

Description: `gs_set_archive_standby_name_on_primary` is used to specify the standby node for archiving. After the setting, the specified standby node

performs the archive operation. The input is the name of a standby node, for example, **dn\_6002**.

Return type: Boolean

- **true**: The setting is successful.
- **false**: The setting fails.

## Restoration Control Functions

Restoration control functions provide information about the status of standby nodes. These functions may be executed both during restoration and in normal running.

- `pg_is_in_recovery()`

Description: Returns **true** if restoration is still in progress.

Return type: Boolean

- `pg_last_xlog_receive_location()`

Description: Obtains the last transaction log location received and synchronized to disk by streaming replication. While streaming replication is in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received and synchronized to disk during restoration. If streaming replication is disabled or if it has not yet started, the function returns a null value.

Return type: text

- `pg_last_xlog_replay_location()`

Description: Obtains last transaction log location replayed during restoration. If restoration is still in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received during that restoration. When the server has been started normally without restoration, the function returns a null value.

Return type: text

- `pg_last_xact_replay_timestamp()`

Description: Obtains the timestamp of last transaction replayed during restoration. This is the time to commit a transaction or abort a WAL record on the primary node. If no transactions have been replayed during restoration, this function will return a null value. If restoration is still in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received during that restoration. If the server normally starts without manual intervention, this function will return a null value.

Return type: timestamp with time zone

Restoration control functions control restoration processes. These functions may be executed only during restoration.

- `pg_is_xlog_replay_paused()`

Description: Returns **true** if restoration is paused.

Return type: Boolean

- `pg_xlog_replay_pause()`  
Description: Pauses restoration immediately.  
Return type: void
- `pg_xlog_replay_resume()`  
Description: Restarts restoration if it was paused.  
Return type: void
- `gs_get_active_archiving_standby()`  
Description: Queries information about archive standby nodes in the same shard. The standby node name, archive location, and number of archived logs are returned.  
Return type: text, int
- `gs_pitr_get_warning_for_xlog_force_recycle()`  
Description: Checks whether logs are recycled because a large number of logs are stacked in the archive slot after archiving is enabled.  
Return type: Boolean
- `gs_pitr_clean_history_global_barriers(stop_barrier_timestamp cstring)`  
Description: Clears all barrier records generated before the specified time. The earliest barrier record is returned. The input parameter is of the cstring type and is a Linux timestamp. You need to perform this operation as an administrator or O&M administrator.  
Return type: text
- `gs_pitr_archive_slot_force_advance(stop_barrier_timestamp cstring)`  
Description: Forcibly updates the archive slot and clears unnecessary barrier records. The new archive slot location is returned. The input parameter is of the cstring type and is a Linux timestamp. You need to perform this operation as an administrator or O&M administrator.  
Return type: text

While restoration is paused, no further database changes are applied. In hot standby mode, all new queries will see the same consistent snapshot of the database, and no further query conflicts will be generated until restoration is resumed.

If streaming replication is disabled, the paused state may continue indefinitely without problem. While streaming replication is in progress, WAL records will continue to be received, which will eventually fill available disk space. This progress depends on the duration of the pause, the rate of WAL generation, and available disk space.

- `gs_recent_barrier_buffer_info(start_time text, end_time text)`  
Description: Queries barrier information based on the time range entered by the user to obtain `time_stamp`, `CSN`, `LSN`, and `standard_time`.  
Return type: records  
Note: To call this function, you must have the `SYSADMIN` or `OPRADMIN` permission. The input parameters **start\_time** and **end\_time** are in the format of year-month-day time, where the time is in the clock format. The maximum query time span is one day. If the time span exceeds the limit, the end time is automatically converted to the limit boundary based on the query start time.  
Example:

```
gaussdb=# SELECT * FROM gs_recent_barrier_buffer_info('2024-01-15 23:27:50', '2024-01-15
23:28:00');
timestamp | lsn | csn | standard_time
-----+-----+-----+-----
1705332470 | 00000000/15FFBBA0 | 41020421 | 2024-01-15 23:27:50
1705332471 | 00000000/15FFBDF0 | 41020422 | 2024-01-15 23:27:51
1705332472 | 00000000/15FFC058 | 41020423 | 2024-01-15 23:27:52
1705332472 | 00000000/15FFC0F8 | 41020424 | 2024-01-15 23:27:52
1705332473 | 00000000/15FFC348 | 41020425 | 2024-01-15 23:27:53
1705332474 | 00000000/15FFC598 | 41020426 | 2024-01-15 23:27:54
1705332475 | 00000000/15FFC638 | 41020427 | 2024-01-15 23:27:55
1705332476 | 00000000/15FFC888 | 41020428 | 2024-01-15 23:27:56
1705332476 | 00000000/15FFDC80 | 41020433 | 2024-01-15 23:27:56
1705332477 | 00000000/15FFDD20 | 41020434 | 2024-01-15 23:27:57
1705332478 | 00000000/15FFDF70 | 41020435 | 2024-01-15 23:27:58
1705332479 | 00000000/15FFE1D8 | 41020436 | 2024-01-15 23:27:59
1705332480 | 00000000/15FFE278 | 41020437 | 2024-01-15 23:28:00
1705332480 | 00000000/15FFE4C8 | 41020438 | 2024-01-15 23:28:00
(14 rows)
```

- `gs_show_obs_media_files(slot_name cstring, src cstring, offset int32, limit int32)`

Description: Queries the OBS file list based on the archive slot (**slot\_name**) and OBS directory address (**src**) entered by the user.

Return type: records

Note: To call this function, you must have the SYSADMIN or OPRADMIN permission. **offset** indicates the query result offset, and **limit** indicates the maximum number of output lines. All files in **src** are queried. Example:

```
gaussdb=# SELECT gs_show_obs_archive_files('ssh','cn_5001/pg_xlog',1, 5);
gs_show_obs_archive_files

(wstdist_ssh/archive/cn_5001/pg_xlog/
0000000100000000000000010_01_01_00000000_00000000_00000003)
(wstdist_ssh/archive/cn_5001/pg_xlog/
0000000100000000000000010_02_01_00000000_00000000_00000003)
(wstdist_ssh/archive/cn_5001/pg_xlog/
0000000100000000000000010_03_01_00000000_00000000_00000003)
(wstdist_ssh/archive/cn_5001/pg_xlog/
0000000100000000000000011_00_01_00000000_00000000_00000003)
(wstdist_ssh/archive/cn_5001/pg_xlog/
0000000100000000000000011_01_01_00000000_00000000_00000003)
(5 rows)
```

- `gs_upload_obs_media_file(slot_name cstring, src cstring, dest cstring, is_forced bool)`

Description: Uploads OBS files based on the archive slot (**slot\_name**), source address (**src**), OBS address (**dest**), and whether to forcibly upload files (**is\_forced**).

Return type: void

Note: To call this function, you must have the SYSADMIN or OPRADMIN permission. The original file directory must be the directory specified by `$GAUSSLOG`. Example:

```
gaussdb=# SELECT * FROM gs_upload_obs_archive_file('ssh', '/data/gauss/log/stwang/test/
0000000100000000000000019_02_01_00000000_00000000_00000003', 'cn_5001/pg_xlog/
0000000100000000000000019_02_01_00000000_00000000_00000003', true);
gs_upload_obs_archive_file

(1 row)
```

- `gs_download_obs_media_file(slot_name cstring, src cstring, dest cstring)`

Description: Downloads OBS files based on the archive slot (**slot\_name**), download source address (**src**), and local destination address (**dest**).

Return type: void

Note: To call this function, you must have the SYSADMIN or OPRADMIN permission. The download directory must be the directory specified by *\$GAUSSLOG*. Example:

```
gaussdb=# SELECT * FROM gs_download_obs_archive_file('ssh','cn_5001/pg_xlog/
000000010000000000000019_02_01_00000000_00000000_00000003','/data/gauss/log/stwang/test');
gs_download_obs_archive_file

(1 row)
```

### 7.5.25.5 Dual-Cluster DR Control Functions

- gs\_streaming\_dr\_in\_switchover()

Description: Truncates services during a planned switchover in the primary cluster in streaming replication-based remote DR solutions.

Return type: Boolean, indicating whether the service truncation is successful and whether the switchover process can be performed normally.

```
gaussdb=# SELECT * FROM gs_streaming_dr_in_switchover();
is_in_switchover

f
(1 row)
```

### 7.5.25.6 Dual-Cluster DR Query Functions

- gs\_get\_local\_barrier\_status()

Description: If two-city 3DC DR is enabled, the primary cluster and standby cluster for DR synchronize logs. The barrier log is flushed to disks in the primary cluster, and replayed in the standby cluster for DR to determine the archive log progress of the primary cluster and the log replay progress of the standby cluster for DR. *gs\_get\_local\_barrier\_status* is used to query the current log replay status of each node in the standby cluster for DR.

Return type: text

**barrier\_id**: latest barrier ID of a node in the standby cluster for DR.

**barrier\_lsn**: LSN of the latest barrier ID returned by a node in the standby cluster for DR.

**archive\_lsn**: location of archived logs obtained by a node in the standby cluster for DR. This parameter does not take effect currently.

**flush\_lsn**: location of logs that have been flushed to disks on a node in the standby cluster for DR.

```
gaussdb=# SELECT * FROM gs_get_local_barrier_status();
barrier_id | barrier_lsn | archive_lsn | flush_lsn
-----+-----+-----+-----
csn_000000000000000028741_1719458320165 | 00000000/2BB4B538 | 00000000/00000000 |
00000000/00000000
(1 row)
```

- gs\_hadr\_in\_recovery()

Description: Checks whether the current node is in barrier-based log restoration if two-city 3DC DR is enabled. If it is in restoration, **true** is returned. Only after the log restoration is complete, can the standby cluster

for DR be promoted to the production cluster during the switchover process. This operation must be performed by system administrators.

Return type: Boolean

```
gaussdb=# SELECT * FROM gs_hadr_in_recovery();
is_in_recovery

t
(1 row)
```

#### NOTE

This function is used only when a planned switchover is performed in the DR cluster.

- `gs_streaming_dr_get_switchover_barrier()`

Description: Checks whether the CN and main standby DN in the standby cluster for DR have received the switchover barrier logs and replayed the logs in the streaming replication-based two-city 3DC DR solution. If it has, **true** is returned. In the standby cluster for DR, the procedure for promoting the DR database instance to the production database instance in the switchover process can be started only after the switchover barrier logs of all DNs are replayed (restricted to system administrators).

Return type: Boolean

Note: This function is used only when a planned switchover is performed in the DR database instance in streaming DR solutions.

```
gaussdb=# SELECT * FROM gs_streaming_dr_get_switchover_barrier();
get_switchover_barrier

f
(1 row)
```

- `gs_streaming_dr_service_truncation_check()`

Description: Checks whether the CN and primary DN in the primary cluster has sent the switchover barrier logs in the streaming replication-based two-city 3DC DR solution. If it has, **true** is returned. The procedure for demoting the production database instance to the DR database instance in the switchover process can be started only after the logs are sent (restricted to system administrators).

Return type: Boolean

Note: This function is used only when a planned switchover is performed in the DR database instance.

```
gaussdb=# SELECT * FROM gs_streaming_dr_service_truncation_check();
complete_truncation

f
(1 row)
```

- `gs_hadr_local_rto_and_rpo_stat()`

Description: Displays the log flow control information of the local database instance and DR database instance for streaming DR. (If this command is executed on a node that does not participate in streaming DR, for example, a standby DN or a CN, no information may be returned.)

The return value type is record. The types and meanings of the columns are as described in [Table 7-63](#).

**Table 7-63** Parameters of gs\_hadr\_local\_rto\_and\_rpo\_stat

| Parameter               | Type | Description                                                                                                                                   |
|-------------------------|------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| hadr_sender_node_name   | text | Node name, including the primary database instance and the main standby node of the standby database instance.                                |
| hadr_receiver_node_name | text | Name of the main standby node of the standby database instance.                                                                               |
| source_ip               | text | IP address of the primary DN of the primary database instance.                                                                                |
| source_port             | int  | Communication port of the primary DN of the primary database instance.                                                                        |
| dest_ip                 | text | IP address of the main standby DN of the standby database instance.                                                                           |
| dest_port               | int  | Communication port of the main standby DN of the standby database instance.                                                                   |
| current_rto             | int  | Flow control information, that is, log RTO time of the current primary and standby database instances (unit: second).                         |
| target_rto              | int  | Flow control information, that is, RTO time between the target primary and standby database instances (unit: second).                         |
| current_rpo             | int  | Flow control information, that is, log RPO time of the current primary and standby database instances (unit: second).                         |
| target_rpo              | int  | Flow control information, that is, RPO time between the target primary and standby database instances (unit: second).                         |
| rto_sleep_time          | int  | RTO flow control information, that is, expected sleep time (unit: $\mu$ s) required by walsender on the host to reach the specified RTO.      |
| rpo_sleep_time          | int  | RPO flow control information, that is, the expected sleep time (unit: $\mu$ s) required by xloginsert on the host to reach the specified RPO. |

```
gaussdb=# SELECT * FROM gs_hadr_local_rto_and_rpo_stat();
hadr_sender_node_name | hadr_receiver_node_name | source_ip | source_port | dest_ip |
dest_port | current_rto | target_rto | current_rpo | target_rpo | rto_sleep_time | rpo_sleep_time
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
dn_6001_6002_6003_6004 | cbg_order_b_hadr_dn_6001 | <IP> | <PORT> | <IP> | <PORT>
| 0 | 0 | 2 | 0 | 0 | 0
(1 row)
```

- gs\_hadr\_remote\_rto\_and\_rpo\_stat()

Description: Displays the log flow control information of all other shards or CN database instances and DR database instances for streaming DR.

(Generally, this command is executed on CNs. If this command is executed on DN, no information may be returned.)

The return value type is record. The types and meanings of the columns are as described in [Table 7-64](#).

**Table 7-64** gs\_hadr\_remote\_rto\_and\_rpo\_stat

| Parameter               | Type | Description                                                                                                                                   |
|-------------------------|------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| hadr_sender_node_name   | text | Node name, including the primary database instance and the main standby node of the standby database instance.                                |
| hadr_receiver_node_name | text | Name of the main standby node of the standby database instance.                                                                               |
| source_ip               | text | IP address of the primary DN of the primary database instance.                                                                                |
| source_port             | int  | Communication port of the primary DN of the primary database instance.                                                                        |
| dest_ip                 | text | IP address of the main standby DN of the standby database instance.                                                                           |
| dest_port               | int  | Communication port of the main standby DN of the standby database instance.                                                                   |
| current_rto             | int  | Flow control information, that is, log RTO time of the current primary and standby database instances (unit: second).                         |
| target_rto              | int  | Flow control information, that is, RTO time between the target primary and standby database instances (unit: second).                         |
| current_rpo             | int  | Flow control information, that is, log RPO time of the current primary and standby database instances (unit: second).                         |
| target_rpo              | int  | Flow control information, that is, RPO time between the target primary and standby database instances (unit: second).                         |
| rto_sleep_time          | int  | RTO flow control information, that is, expected sleep time (unit: $\mu$ s) required by walsender on the host to reach the specified RTO.      |
| rpo_sleep_time          | int  | RPO flow control information, that is, the expected sleep time (unit: $\mu$ s) required by xlogInsert on the host to reach the specified RPO. |

```
gaussdb=# SELECT * FROM gs_hadr_remote_rto_and_rpo_stat();
hadr_sender_node_name | hadr_receiver_node_name | source_ip | source_port | dest_ip |
dest_port | current_rto | target_rto | current_rpo | target_rpo | rto_sleep_time | rpo_sleep_time
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
dn_6001_6002_6003_6004 | cbg_order_b_hadr_dn_6001 | <IP> | <PORT> | <IP> | <PORT>
| 0 | 0 | 4 | 0 | 0 | 0
dn_6013_6014_6015_6016 | cbg_order_b_hadr_dn_6007 | <IP> | <PORT> | <IP> | <PORT>
| 0 | 0 | 2 | 0 | 0 | 0
dn_6009_6010_6011_6012 | cbg_order_b_hadr_dn_6005 | <IP> | <PORT> | <IP> | <PORT>
| 0 | 0 | 2 | 0 | 0 | 0
dn_6005_6006_6007_6008 | cbg_order_b_hadr_dn_6003 | <IP> | <PORT> | <IP> | <PORT>
| 0 | 0 | 4 | 0 | 0 | 0
cn_5002 | cbg_order_b_hadr_cn_5002 | <IP> | <PORT> | <IP> | <PORT> | 0
| 0 | 1 | 0 | 0 | 0
(5 rows)

```

### 7.5.25.7 Snapshot Synchronization Functions

Snapshot synchronization functions save the current snapshot and return its identifier.

- `pg_export_snapshot()`

Description: Saves the current snapshot and returns its identifier.

Return type: text

Note: `pg_export_snapshot` saves the current snapshot and returns a text string identifying the snapshot. This string must be passed to clients that want to import the snapshot. A snapshot can be imported when the **set transaction snapshot snapshot\_id;** command is executed. Doing so is possible only when the transaction is set to the **SERIALIZABLE** or **REPEATABLE READ** isolation level. GaussDB does not support these two isolation levels currently. The output of the function cannot be used as the input of **set transaction snapshot.**

```

gaussdb=# SELECT * FROM pg_export_snapshot();
pg_export_snapshot

0000000000002D7E8-1
(1 row)

```

- `pg_export_snapshot_and_csn()`

Description: Saves the current snapshot and returns its identifier. Compared with `pg_export_snapshot()`, it returns a CSN, indicating the CSN of the current snapshot.

Return type: text

```

gaussdb=# SELECT * FROM pg_export_snapshot_and_csn();
 snapshot | csn
-----+-----
0000000000002D7DE-1 | 5E4A6E
(1 row)

```

### 7.5.25.8 Database Object Functions

#### Database Object Size Functions

Database object size functions calculate the actual disk space used by database objects.

- `pg_column_size(any)`

Description: Specifies the number of bytes used to store a particular value (possibly compressed)

Return type: int

Note: `pg_column_size` displays the space for storing an independent data value.

```
gaussdb=# SELECT pg_column_size(1);
pg_column_size

 4
(1 row)
```

- `pg_database_size(oid)`

Description: Specifies the disk space used by the database with the specified OID.

Return type: `bigint`

- `pg_database_size(name)`

Description: Specifies the disk space used by the database with the specified name.

Return type: `bigint`

Note: `pg_database_size` receives the OID or name of a database and returns the disk space used by the corresponding object.

Example:

```
gaussdb=# CREATE DATABASE testdb dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# SELECT pg_database_size('testdb');
pg_database_size

 51590112
(1 row)
gaussdb=# DROP DATABASE testdb;
DROP DATABASE
```

- `pg_relation_size(oid)`

Description: Specifies the disk space used by the table with a specified OID or index.

Return type: `bigint`

- `get_db_source_datasize()`

Description: Estimates the total size of non-compressed data in the current database.

Return type: `bigint`

Note: Perform an analysis before this function is called.

Example:

```
gaussdb=# analyze;
ANALYZE
gaussdb=# SELECT get_db_source_datasize();
get_db_source_datasize

 35384925667
(1 row)
```

- `pg_relation_size(text)`

Description: Specifies the disk space used by the table with a specified name or index. The table name can be schema-qualified.

Return type: `bigint`

- `pg_relation_size(relation regclass, fork text)`

Description: Specifies the disk space used by the specified bifurcating tree ('main', 'fsm', or 'vm') of a certain table or index.

- Return type: bigint
- `pg_relation_size(regclass)`  
Description: Is an abbreviation of `pg_relation_size(..., 'main')`.  
Return type: bigint  
Note: `pg_relation_size` receives the OID or name of a table, an index, or a compressed table, and returns the size.
  - `pg_partition_size(oid, oid)`  
Description: Specifies the disk space used by the partition with a specified OID. The first **oid** is the OID of the table and the second **oid** is the OID of the partition.  
Return type: bigint
  - `pg_partition_size(text, text)`  
Description: Specifies the disk space used by the partition with a specified name. The first **text** is the table name and the second **text** is the partition name.  
Return type: bigint
  - `pg_partition_indexes_size(oid, oid)`  
Description: Specifies the disk space used by the index of the partition with a specified OID. The first **oid** is the OID of the table and the second **oid** is the OID of the partition.  
Return type: bigint
  - `pg_partition_indexes_size(text, text)`  
Description: Specifies the disk space used by the index of the partition with a specified name. The first **text** is the table name and the second **text** is the partition name.  
Return type: bigint
  - `pg_indexes_size(regclass)`  
Description: Specifies the total disk space used by the index appended to the specified table.  
Return type: bigint
  - `pg_size_pretty(bigint)`  
Description: Converts a size in bytes expressed as a 64-bit integer into a human-readable format with size units.  
Return type: text
  - `pg_size_pretty(numeric)`  
Description: Converts a size in bytes expressed as a numeric value into a human-readable format with size units.  
Return type: text  
Note: `pg_size_pretty` formats the results of other functions into a human-readable format. KB, MB, GB, and TB can be used.
  - `pg_table_size(regclass)`  
Description: Specifies the disk space used by the specified table, excluding indexes (but including TOAST, free space mapping, and visibility mapping).  
Return type: bigint

- pg\_tablespace\_size(oid)**  
Description: Specifies the disk space used by the tablespace with a specified OID.  
Return type: bigint
- pg\_tablespace\_size(name)**  
Description: Specifies the disk space used by the tablespace with a specified name.  
Return type: bigint  
Note:  
pg\_tablespace\_size receives the OID or name of a database and returns the disk space used by the corresponding object.
- pg\_total\_relation\_size(oid)**  
Description: Specifies the disk space used by the table with a specified OID, including the index and the compressed data.  
Return type: bigint
- pg\_total\_relation\_size(regclass)**  
Description: Specifies the total disk space used by the specified table, including all indexes and TOAST data.  
Return type: bigint
- pg\_total\_relation\_size(text)**  
Description: Specifies the disk space used by the table with a specified name, including the index and the compressed data. The table name can be schema-qualified.  
Return type: bigint  
Note: pg\_total\_relation\_size receives the OID or name of a table or a compressed table, and returns the sizes of the data, related indexes, and the compressed table in bytes.
- datalength(any)**  
Description: Specifies the number of bytes used by an expression of a specified data type (data management space, data compression, or data type conversion is not considered).  
Return type: int  
Note: datalength calculates the space of an independent data value.

Example:

```
gaussdb=# SELECT datalength(1);
datalength

4
(1 row)
```

The following table lists the supported data types and calculation methods.

| Data Type          |               |          | Storage Space |
|--------------------|---------------|----------|---------------|
| Numeric data types | Integer types | TINYINT  | 1             |
|                    |               | SMALLINT | 2             |

| Data Type |                           | Storage Space    |                                                                                                                      |
|-----------|---------------------------|------------------|----------------------------------------------------------------------------------------------------------------------|
|           |                           | INTEGER          | 4                                                                                                                    |
|           |                           | BINARY_INTEGER   | 4                                                                                                                    |
|           |                           | BIGINT           | 8                                                                                                                    |
|           | Arbitrary precision types | DECIMAL          | Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately. |
|           |                           | NUMERIC          | Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately. |
|           |                           | NUMBER           | Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately. |
|           | Sequence integer          | SMALLSERIAL      | 2                                                                                                                    |
|           |                           | SERIAL           | 4                                                                                                                    |
|           |                           | BIGSERIAL        | 8                                                                                                                    |
|           | Floating-point types      | FLOAT4           | 4                                                                                                                    |
|           |                           | DOUBLE PRECISION | 8                                                                                                                    |
|           |                           | FLOAT8           | 8                                                                                                                    |
|           |                           | BINARY_DOUBLE    | 8                                                                                                                    |
|           |                           | FLOAT[(p)]       | Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately. |
|           |                           | DEC[(p,s)]       | Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately. |
|           |                           | INTEGER[(p,s)]   | Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately. |

| Data Type            |                 |                        | Storage Space                          |
|----------------------|-----------------|------------------------|----------------------------------------|
| Boolean data types   | Boolean type    | BOOLEAN                | 1                                      |
| Character data types | Character types | CHAR                   | n                                      |
|                      |                 | CHAR(n)                | n                                      |
|                      |                 | CHARACTER(n)           | n                                      |
|                      |                 | NCHAR(n)               | n                                      |
|                      |                 | VARCHAR(n)             | n                                      |
|                      |                 | CHARACTER              | Actual number of bytes of a character. |
|                      |                 | VARYING(n)             | Actual number of bytes of a character. |
|                      |                 | VARCHAR2(n)            | Actual number of bytes of a character. |
|                      |                 | NVARCHAR2(n)           | Actual number of bytes of a character. |
|                      |                 | TEXT                   | Actual number of bytes of a character. |
|                      |                 | CLOB                   | Actual number of bytes of a character. |
| Time data types      | Time types      | DATE                   | 8                                      |
|                      |                 | TIME                   | 8                                      |
|                      |                 | TIMEZ                  | 12                                     |
|                      |                 | TIMESTAMP              | 8                                      |
|                      |                 | TIMESTAMPZ             | 8                                      |
|                      |                 | SMALLDATETIME          | 8                                      |
|                      |                 | INTERVAL DAY TO SECOND | 16                                     |
|                      |                 | INTERVAL               | 16                                     |
|                      |                 | RELTIME                | 4                                      |
|                      |                 | ABSTIME                | 4                                      |
|                      |                 | TINTERVAL              | 12                                     |

## Database Object Position Functions

- `pg_relation_filenode(relation regclass)`  
Description: Specifies the ID of a filenode with the specified relationship.  
Return type: oid  
Note: `pg_relation_filenode` receives the OID or name of a table, an index, a sequence, or a compressed table, and returns the ID of **filenode** allocated to it. **filenode** is the basic component of the file name used by the relationship. For most tables, the result is the same as that of **pg\_class.relfilenode**. For the specified system directory, **relfilenode** is **0** and this function must be used to obtain the correct value. If a relationship that is not stored is transmitted, such as a view, this function returns a null value.
- `pg_relation_filepath(relation regclass)`  
Description: Specifies the name of a file path with the specified relationship.  
Return type: text  
Description: `pg_relation_filepath` is similar to `pg_relation_filenode`, except that `pg_relation_filepath` returns the whole file path name for the relationship (relative to the data directory **PGDATA** of the database cluster).
- `get_large_table_name(relfile_node text, threshold_size_gb int8)`  
Description: Queries whether the table size (in GB) exceeds the threshold (**threshold\_size\_gb**) based on the table file code (**relfile\_node**). If the size exceeds the threshold, the schema name and table name (in *schemaname.tablename* format) are returned, otherwise, **null** is returned.  
Return type: text
- `pg_filenode_relation tablespacename, relname)`  
Description: Obtains the table names corresponding to the **tablespace** and **relfilenode**.  
Return type: regclass
- `pg_partition_filenode(partition_oid)`  
Description: Obtains **filenode** corresponding to the OID lock of a specified partitioned table.  
Return type: oid
- `pg_partition_filepath(partition_oid)`  
Description: Specifies the file path name of a partition.  
Return type: text

## Recycle Bin Object Functions

- `gs_is_recycle_object(classid, objid, objname)`  
Description: Determines whether an object is in the recycle bin. This function is not supported in distributed mode.  
Return type: Boolean

### 7.5.25.9 Advisory Lock Functions

Advisory lock functions manage advisory locks.

- `pg_advisory_lock(key bigint)`  
Description: Obtains an exclusive session-level advisory lock.  
Return type: void  
Note: `pg_advisory_lock` locks resources defined by an application. The resources can be identified using a 64-bit or two nonoverlapped 32-bit key values. If another session locks the resources, the function blocks the resources until they can be used. The lock is exclusive. Multiple locking requests are pushed into the stack. Therefore, if the same resource is locked three times, it must be unlocked three times so that it is released to another session.
- `pg_advisory_lock(key1 int, key2 int)`  
Description: Obtains an exclusive session-level advisory lock.  
Return type: void  
Note: Only users with the SYSADMIN permission can add session-level exclusive advisory locks to the key-value pair (65535, 65535).
- `pg_advisory_lock(lock_id int4, lock_id int4, database_name Name)`  
Description: Obtains the exclusive advisory lock of a specified database by inputting the lock ID and database name.  
Return type: void
- `pg_advisory_lock_shared(key bigint)`  
Description: Obtains a shared session-level advisory lock.  
Return type: void
- `pg_advisory_lock_shared(key1 int, key2 int)`  
Description: Obtains a shared session-level advisory lock.  
Return type: void  
Note: `pg_advisory_lock_shared` works in the same way as `pg_advisory_lock`, except that `pg_advisory_lock_shared` obtains an advisory lock shared with other sessions requesting the lock, while `pg_advisory_lock` obtains an exclusive advisory lock.
- `pg_advisory_unlock(key bigint)`  
Description: Releases an exclusive session-level advisory lock.  
Return type: Boolean
- `pg_advisory_unlock(key1 int, key2 int)`  
Description: Releases an exclusive session-level advisory lock.  
Return type: Boolean  
Note: `pg_advisory_unlock` releases the obtained exclusive advisory lock. If the release is successful, the function returns **true**. If the lock was not held, it will return **false**. In addition, an SQL warning will be reported by the server.
- `pg_advisory_unlock(lock_id int4, lock_id int4, database_name Name)`  
Description: Releases an exclusive advisory lock of a specified database by inputting the lock ID and database name.  
Return type: Boolean  
Note: If the release is successful, **true** is returned. If no lock is held, **false** is returned.

- `pg_advisory_unlock_shared(key bigint)`  
Description: Releases a shared session-level advisory lock.  
Return type: Boolean
- `pg_advisory_unlock_shared(key1 int, key2 int)`  
Description: Releases a shared session-level advisory lock.  
Return type: Boolean  
Note: `pg_advisory_unlock_shared` works in the same way as `pg_advisory_unlock`, except it releases a shared session-level advisory lock.
- `pg_advisory_unlock_all()`  
Description: Releases all advisory locks owned by the current session.  
Return type: void  
Note: `pg_advisory_unlock_all` releases all advisory locks owned by the current session. The function is implicitly called when the session ends even if the client is abnormally disconnected.
- `pg_advisory_xact_lock(key bigint)`  
Description: Obtains an exclusive transaction-level advisory lock.  
Return type: void
- `pg_advisory_xact_lock(key1 int, key2 int)`  
Description: Obtains an exclusive transaction-level advisory lock.  
Return type: void  
Note: `pg_advisory_xact_lock` works in the same way as `pg_advisory_lock`, except that the lock is automatically released at the end of the current transaction and cannot be released explicitly. Only users with the SYSADMIN permission can add transaction-level exclusive advisory locks to the key-value pair (65535, 65535).
- `pg_advisory_xact_lock_shared(key bigint)`  
Description: Obtains a shared transaction-level advisory lock.  
Return type: void
- `pg_advisory_xact_lock_shared(key1 int, key2 int)`  
Description: Obtains a shared transaction-level advisory lock.  
Return type: void  
Note: `pg_advisory_xact_lock_shared` works in the same way as `pg_advisory_lock_shared`, except the lock is automatically released at the end of the current transaction and cannot be released explicitly.
- `pg_try_advisory_lock(key bigint)`  
Description: Obtains an exclusive session-level advisory lock if available.  
Return type: Boolean  
Note: `pg_try_advisory_lock` works in the same way as `pg_advisory_lock`, except `pg_try_advisory_lock` does not block the resource until the resource is released. It either immediately obtains the lock and returns **true** or returns **false**, which indicates the lock cannot be performed currently.
- `pg_try_advisory_lock(key1 int, key2 int)`  
Description: Obtains an exclusive session-level advisory lock if available.  
Return type: Boolean

Note: Only users with the SYSADMIN permission can add session-level exclusive advisory locks to the key-value pair (65535, 65535).

- `pg_try_advisory_lock_shared(key bigint)`  
Description: Obtains a shared session-level advisory lock if available.  
Return type: Boolean
- `pg_try_advisory_lock_shared(key1 int, key2 int)`  
Description: Obtains a shared session-level advisory lock if available.  
Return type: Boolean  
Note: `pg_try_advisory_lock_shared` works in the same way as `pg_try_advisory_lock`, except that `pg_try_advisory_lock_shared` attempts to obtain a shared lock instead of an exclusive lock.
- `pg_try_advisory_xact_lock(key bigint)`  
Description: Obtains an exclusive transaction-level advisory lock if available.  
Return type: Boolean
- `pg_try_advisory_xact_lock(key1 int, key2 int)`  
Description: Obtains an exclusive transaction-level advisory lock if available.  
Return type: Boolean  
Note: **`pg_try_advisory_xact_lock`** works in the same way as **`pg_try_advisory_lock`**, except that the lock, if acquired, is automatically released at the end of the current transaction and cannot be released explicitly. Only users with the SYSADMIN permission can add transaction-level exclusive advisory locks to the key-value pair (65535, 65535).
- `pg_try_advisory_xact_lock_shared(key bigint)`  
Description: Obtains a shared transaction-level advisory lock if available.  
Return type: Boolean
- `pg_try_advisory_xact_lock_shared(key1 int, key2 int)`  
Description: Obtains a shared transaction-level advisory lock if available.  
Return type: Boolean  
Note: `pg_try_advisory_xact_lock_shared` works in the same way as `pg_try_advisory_lock_shared`, except the lock, if acquired, is automatically released at the end of the current transaction and cannot be released explicitly.
- `lock_cluster_ddl()`  
Description: Attempts to obtain a session-level exclusive advisory lock for all active CNs in the cluster.  
Return type: Boolean  
Note: Only users with the SYSADMIN permission can call this function.
- `unlock_cluster_ddl()`  
Description: Attempts to add a session-level exclusive advisory lock on a CN.  
Return type: Boolean

Example of session-level advisory locks:

```
gaussdb=# select pg_advisory_lock(1); -- Obtain three session-level advisory locks in sequence.
pg_advisory_lock

```







```

t
(1 row)

gaussdb=# select pg_advisory_unlock_all(); -- Release the advisory lock.
pg_advisory_unlock_all

(1 row)

gaussdb=# select pg_advisory_lock(1); -- Obtain the exclusive advisory lock on object 1.
pg_advisory_lock

(1 row)

-- Keep the connection and start a new connection.
gaussdb=# select pg_try_advisory_lock_shared(1); -- Failed to obtain the shared advisory lock on
object 1 in a new connection because an existing connection holds an exclusive lock.
pg_try_advisory_lock_shared

f
(1 row)
```

### 7.5.25.10 Logical Replication Functions

#### NOTE

When using the logical replication functions, set the GUC parameter **wal\_level** to **logical**. For details about the configuration, see "Logical Replication > Logical Decoding > Logical Decoding by SQL Function Interfaces" in *Feature Guide*.

- `pg_create_logical_replication_slot('slot_name', 'plugin_name', 'output_order')`

Description: Creates a logical replication slot.

Parameters:

- `slot_name`

Indicates the name of the streaming replication slot.

Value range: a string, supporting only lowercase letters, digits, underscores (`_`), question marks (`?`), hyphens (`-`), and periods (`.`). One or two periods cannot be used alone as the replication slot name.

- `plugin_name`

Indicates the name of the plug-in.

Value range: a string, supporting **mppdb\_decoding**

- `output_order`

Indicates the output sequence of the replication slot decoding results. This parameter is optional.

Valid value: **0** or **1**. The default value is **0** on DNs and **1** on CNs.

- **0**: The replication slot decoding results are sorted by transaction COMMIT LSN. In this case, the value of **confirmed\_csn** of the replication slot is **0**. This replication slot is called an LSN-based replication slot.
- **1**: The replication slot decoding results are sorted by transaction CSN. In this case, the value of **confirmed\_csn** of the replication slot is not **0**. This replication slot is called a CSN-based replication slot. In this case, the **confirmed\_flush** value is meaningless.

Return type: name, text

Example:

```
gaussdb=# SELECT * FROM pg_create_logical_replication_slot('slot_lsn','mppdb_decoding',0);
 slotname | xlog_position
```

```
-----+-----
slot_lsn | 0/6D08B58
(1 row)
```

```
gaussdb=# SELECT * FROM pg_create_logical_replication_slot('slot_csn','mppdb_decoding',1);
 slotname | xlog_position
```

```
-----+-----
slot_csn | 0/59AD800
(1 row)
```

Note: The first return value is the slot name, and the second one has different meanings in LSN-based replication slots and CSN-based replication slots. For an LSN-based replication slot, the value is **confirmed\_flush** of the replication slot, indicating that transactions whose commit LSN is less than or equal to the value will not be decoded and output. For a CSN-based replication slot, the value is **confirmed\_csn** of the replication slot, indicating that transactions whose CSN is less than or equal to the value will not be decoded and output. Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit permissions of the built-in role `gs_role_replication`. Currently, this function can be called only on the CN or primary DNs.

---

 **CAUTION**

When this function is executed on a CN, if **output\_order** is set to **1**, a replication slot with the same name will be created on all CNs and primary DNs. If a replication slot with the same name already exists on some DNs, a CSN-based logical replication slot will be created on other CNs and primary DNs that do not have a replication slot with the same name. In addition, an error message is displayed, indicating that the replication slot already exists. In this case, you need to delete the existing replication slot with the same name on the DN (if the replication slot is an LSN-based logical replication slot, you need to manually delete it on the corresponding node) and create a replication slot on the CN. If you set **output\_order** to **0** when creating a replication slot on a CN, no replication slot will be created on DNs. Replication slots created on CNs are only used to identify whether related replication slots exist on DNs.

- 
- `pg_create_physical_replication_slot('slot_name', 'isDummyStandby')`

Description: Creates a physical replication slot.

Parameters:

- `slot_name`

Indicates the name of the streaming replication slot.

Value range: a string, supporting only lowercase letters, digits, underscores (`_`), question marks (`?`), hyphens (`-`), and periods (`.`). One or two periods cannot be used alone as the replication slot name.

- `isDummyStandby`

Reserved parameter.

Type: Boolean

Return type: name, text

Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit permissions of the built-in role `gs_role_replication`. The physical replication slot created by this function does not have `restart_lsn`. Therefore, the slot is considered invalid and will be automatically deleted when the checkpoint is performed.

- `pg_drop_replication_slot('slot_name')`

Description: Deletes a streaming replication slot.

Parameters:

- `slot_name`

Indicates the name of the streaming replication slot.

Value range: a string, supporting only lowercase letters, digits, underscores (`_`), question marks (`?`), hyphens (`-`), and periods (`.`). One or two periods cannot be used alone as the replication slot name.

Return type: void

Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit permissions of the built-in role `gs_role_replication`.

---

 CAUTION

- When a logical replication slot is deleted from a CN, if the logical replication slot is an LSN-based logical replication slot, only the replication slot of the current node is deleted. Replication slots with the same name on other nodes are not affected. When a CSN-based logical replication slot with the same name exists on other nodes, no error is reported because some nodes do not have replication slots. In addition, replication slots with the same name on all nodes are successfully deleted. If no replication slot exists on any node, an error is reported.
- If an LSN-based logical replication slot remains on the current CN and a CSN-based logical replication slot with the same name remains on other nodes, deleting the replication slot on the current CN will delete only the local LSN-based logical replication slot. After the deletion is complete, perform the deletion operation again to delete the CSN-based logical replication slots with the same name on other nodes.

- 
- `pg_logical_slot_peek_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')`

Description: Performs decoding on the decoding DN but does not go to the next streaming replication slot. (The decoded result will be returned again during the next decoding.)

Parameters:

- `slot_name`

Indicates the name of the streaming replication slot.

Value range: a string, supporting only lowercase letters, digits, underscores (`_`), question marks (`?`), hyphens (`-`), and periods (`.`). One or two periods cannot be used alone as the replication slot name.

- upto\_lsn  
For the CSN-based logical replication slot, the decoding is complete until the transaction whose CSN is less than or equal to the value is decoded (a transaction whose CSN is greater than the specified CSN may be decoded). For the LSN-based replication slot, the decoding is complete until the first transaction whose COMMIT LSN is greater than or equal to the value is decoded.

Value range: a string, for example, '1/2AAFC60', '0/A060', or '3A/0' (a hexadecimal uint64 value containing two uint32 values separated by a slash (/); if any uint32 value is **0**, **0** is displayed.) **NULL** indicates that the end position of decoding is not specified.

- upto\_nchanges  
Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto\_nchanges** is set to **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto\_nchanges** after decoding in the first two transactions is finished.

Value range: a non-negative integer

 **NOTE**

If any of the **upto\_lsn** and **upto\_nchanges** values is reached, decoding ends.

- **options**: Specifies optional parameters, consisting of **options\_name** and **options\_value**.

- include-xids

Specifies whether the decoded **data** column contains XID information.

Value range: **0** and **1**. The default value is **1**.

- **0**: The decoded **data** column does not contain XID information.
- **1**: The decoded **data** column contains XID information.

- skip-empty-xacts

Specifies whether to ignore empty transaction information during decoding.

Value range: **0** and **1**. The default value is **0**.

- **0**: The empty transaction information is not ignored during decoding.
- **1**: The empty transaction information is ignored during decoding.

- include-timestamp

Specifies whether decoded information contains the **commit** timestamp.

Value range: **0** and **1**. The default value is **0**.

- **0**: The decoded information does not contain the **commit** timestamp.

- **1**: The decoded information contains the **commit** timestamp.
- **only-local**  
Specifies whether to decode only local logs.  
Value range: **0** and **1**. The default value is **1**.
  - **0**: Non-local logs and local logs are decoded.
  - **1**: Only local logs are decoded.
- **force-binary**  
Specifies whether to output the decoding result in binary format.  
Value range: **0**
  - **0**: The decoding result is output in text format.
- **white-table-list**  
Whitelist parameter, including the schema and table name to be decoded.  
Value range: a string that contains table names in the whitelist. Different tables are separated by commas (.). An asterisk (\*) is used to fuzzily match all tables. Schema names and table names are separated by periods (.). No space character is allowed. The following is an example:

```
gaussdb=# SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');
```
- **max-txn-in-memory**  
Memory control parameter. The unit is MB. If the memory occupied by a single transaction is greater than the value of this parameter, data is flushed to disk.  
Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.
- **max-reorderbuffer-in-memory**  
Memory control parameter. The unit is GB. If the total memory (including the cache) of transactions being concatenated in the sender thread is greater than the value of this parameter, the current decoding transaction is flushed to disk.  
Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.
- **include-user**  
Specifies whether the BEGIN logical log of a transaction records the username of the transaction.  
Value range: **0** and **1**. The default value is **0**.
  - **0**: The BEGIN logical log of a transaction does not contain the username of the transaction.
  - **1**: The BEGIN logical log of a transaction records the username of the transaction.
- **exclude-userids**  
Specifies the OID of a blacklisted user.

Value range: OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not check whether the OIDs exist.

- **exclude-users**

Specifies the name of a blacklisted user.

Value range: names of blacklisted users. Multiple names are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify usernames. If the decoding is interrupted because the user does not exist and the corresponding blacklisted user does not exist at the time when logs are generated, you can set **dynamic-resolution** to **true** or delete the username from the blacklist to start decoding and continue to obtain logical logs.

- **dynamic-resolution**

Specifies whether to dynamically parse the names of blacklisted users.

Value range: **0** and **1**. The default value is **1**.

- **0**: If the parameter is set to **0**, an error is reported and the logical decoding exits when the decoding detects that the user does not exist in blacklist **exclude-users**.
- **1**: If the parameter is set to **1**, decoding continues when it detects that the user does not exist in blacklist **exclude-users**.

 **NOTE**

For details about other configuration options, see "Logical Replication > Logical Decoding > Logical Decoding Options" in *Feature Guide*.

Return type: text, xid, text

Example:

```
gaussdb=# EXECUTE DIRECT ON (datanode1) 'SELECT * FROM
pg_logical_slot_peek_changes("slot_lsn",NULL,4096,"skip-empty-xacts","on");
location | xid | data
-----+-----
-----+-----
0/6D0B500 | 46914 | BEGIN 46914
0/6D0B530 | 46914 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","1"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
0/6D0B8B8 | 46914 | COMMIT 46914 (at 2023-02-22 17:29:31.090018+08) CSN 94034528
0/6D0BB58 | 46915 | BEGIN 46915
0/6D0BB88 | 46915 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","2"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
0/6D0BF08 | 46915 | COMMIT 46915 (at 2023-02-22 17:31:30.672093+08) CSN 94034568
0/6D0BF08 | 46916 | BEGIN 46916
0/6D0BF38 | 46916 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
0/6D0C218 | 46916 | COMMIT 46916 (at 2023-02-22 17:31:34.438319+08) CSN 94034570
(9 rows)

gaussdb=# EXECUTE DIRECT ON (datanode1) 'SELECT * FROM
pg_logical_slot_peek_changes("slot_csn",NULL,4096,"skip-empty-xacts","on");
location | xid | data
-----+-----
-----+-----
```

```

0/0 | 46914 | BEGIN CSN: 94034528
0/0 | 46914 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","1"],"old_keys_name":
[],"old_keys_type":["integer"],"old_keys_val":[]}
0/59ADA60 | 46914 | COMMIT 46914 (at 2023-02-22 17:29:31.090018+08) CSN 94034528
0/59ADA60 | 46915 | BEGIN CSN: 94034568
0/59ADA60 | 46915 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","2"],"old_keys_name":
[],"old_keys_type":["integer"],"old_keys_val":[]}
0/59ADA88 | 46915 | COMMIT 46915 (at 2023-02-22 17:31:30.672093+08) CSN 94034568
0/59ADA88 | 46916 | BEGIN CSN: 94034570
0/59ADA88 | 46916 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":
[],"old_keys_type":["integer"],"old_keys_val":[]}
0/59ADA8A | 46916 | COMMIT 46916 (at 2023-02-22 17:31:34.438319+08) CSN 94034570
(9 rows)

```

Note: In the preceding function example, **datanode1** is the name of the DN to be decoded, and **slot\_lsn** and **slot\_csn** are the names of the logical replication slots. The decoding result returned by the function contains three columns, corresponding to the preceding return value types, which are the LSN (for an LSN-based replication slot) or CSN (for a CSN-based replication slot), XID, and decoded content, respectively. If the **location** column indicates the CSN, the value of the **location** column is updated only when the commit logs are decoded. Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit permissions of the built-in role `gs_role_replication`.

#### NOTE

On the CN, this function cannot be executed in a CSN-based replication slot (**confirmed\_csn** is not 0).

- `pg_logical_slot_get_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')`

Description: Performs decoding on the decoding DN and updates the streaming replication slot.

Parameter: This function has the same parameters as **pg\_logical\_slot\_peek\_changes**. For details, see [•pg\\_logical\\_slot\\_peek\\_ch...](#)

Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit permissions of the built-in role `gs_role_replication`.

#### NOTE

On the CN, this function cannot be executed in a CSN-based replication slot (**confirmed\_csn** is not 0). When this function is called on the standby DN, the corresponding logical replication slot on the primary DN is updated synchronously. If this function is executed on the standby DN, a WAL sender of the primary DN is occupied when the replication slot number on the primary DN is updated. The logical decoding function reserves a WAL sender for each logical replication slot. Therefore, if this function is executed in normal scenarios, the logical replication slot number on the primary DN is updated normally. If this function is executed continuously in a short period of time, the primary DN fails to update the slot number and no error is reported.

- `pg_logical_slot_peek_binary_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')`

Description: Performs decoding in binary format on the decoding DN and does not update the streaming replication slot. (The decoded data can be obtained again during the next decoding.)

Parameters:

- slot\_name

Indicates the name of the streaming replication slot.

Value range: a string, supporting only lowercase letters, digits, underscores (\_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

- upto\_lsn

For the CSN-based logical replication slot, the decoding is complete until the transaction whose CSN is less than or equal to the value is decoded (a transaction whose CSN is greater than the specified CSN may be decoded). For the LSN-based replication slot, the decoding is complete until the first transaction whose COMMIT LSN is greater than or equal to the value is decoded.

Value range: a string, for example, '1/2AAFC60', '0/A060', or '3A/0' (a hexadecimal uint64 value containing two uint32 values separated by a slash (/); if any uint32 value is 0, 0 is displayed.) **NULL** indicates that the end position of decoding is not specified.

- upto\_nchanges

Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto\_nchanges** is set to 4, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto\_nchanges** after decoding in the first two transactions is finished.

Value range: a non-negative integer

 **NOTE**

If any of the **upto\_lsn** and **upto\_nchanges** values is reached, decoding ends.

- **options**: Specifies optional parameters, consisting of **options\_name** and **options\_value**.

▪ include-xids

Specifies whether the decoded **data** column contains XID information.

Value range: **0** and **1**. The default value is **1**.

- **0**: The decoded **data** column does not contain XID information.
- **1**: The decoded **data** column contains XID information.

▪ skip-empty-xacts

Specifies whether to ignore empty transaction information during decoding.

Value range: **0** and **1**. The default value is **0**.

- **0**: The empty transaction information is not ignored during decoding.
- **1**: The empty transaction information is ignored during decoding.

- **include-timestamp**  
Specifies whether decoded information contains the **commit** timestamp.  
Value range: **0** and **1**. The default value is **0**.
  - **0**: The decoded information does not contain the **commit** timestamp.
  - **1**: The decoded information contains the **commit** timestamp.
- **only-local**  
Specifies whether to decode only local logs.  
Value range: **0** and **1**. The default value is **1**.
  - **0**: Non-local logs and local logs are decoded.
  - **1**: Only local logs are decoded.
- **force-binary**  
Specifies whether to output the decoding result in binary format.  
Value range: **0** or **1**. The default value is **0**. The result is output in binary format.
- **white-table-list**  
Whitelist parameter, including the schema and table name to be decoded.  
Value range: a string that contains table names in the whitelist. Different tables are separated by commas (.). An asterisk (\*) is used to fuzzily match all tables. Schema names and table names are separated by periods (.). No space character is allowed. Example:  
**select \* from pg\_logical\_slot\_peek\_binary\_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');**
- **max-txn-in-memory**  
Memory control parameter. The unit is MB. If the memory occupied by a single transaction is greater than the value of this parameter, data is flushed to disk.  
Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.
- **max-reorderbuffer-in-memory**  
Memory control parameter. The unit is GB. If the total memory (including the cache) of transactions being concatenated in the sender thread is greater than the value of this parameter, the current decoding transaction is flushed to disk.  
Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.
- **include-user**  
Specifies whether the BEGIN logical log of a transaction records the username of the transaction.  
Value range: **0** and **1**. The default value is **0**.

- **0**: The BEGIN logical log of a transaction does not contain the username of the transaction.
- **1**: The BEGIN logical log of a transaction records the username of the transaction.
- **exclude-userids**  
Specifies the OID of a blacklisted user.  
Value range: OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not check whether the OIDs exist.
- **exclude-users**  
Specifies the name of a blacklisted user.  
Value range: names of blacklisted users. Multiple names are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify usernames. If the decoding is interrupted because the user does not exist and the corresponding blacklisted user does not exist at the time when logs are generated, you can set **dynamic-resolution** to **true** or delete the username from the blacklist to start decoding and continue to obtain logical logs.
- **dynamic-resolution**  
Specifies whether to dynamically parse the names of blacklisted users.  
Value range: **0** and **1**. The default value is **1**.
  - **0**: If the parameter is set to **0**, an error is reported and the logical decoding exits when the decoding detects that the user does not exist in blacklist **exclude-users**.
  - **1**: If the parameter is set to **1**, decoding continues when it detects that the user does not exist in blacklist **exclude-users**.

 **NOTE**

Some configuration options do not take effect even if they are configured in functions. For details, see "Logical Replication > Logical Decoding > Logical Decoding Options" in *Feature Guide*.

Return type: text, xid, bytea

Note: The function returns the decoding result. Each decoding result contains three columns, corresponding to the above return types and indicating the LSN position, XID, and decoded content in binary format, respectively. Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit permissions of the built-in role `gs_role_replication`.

 **NOTE**

On the CN, this function cannot be executed in a CSN-based replication slot (**confirmed\_csn** is not **0**).

- `pg_logical_slot_get_binary_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')`

Description: Performs decoding in binary format on the decoding DN and updates the streaming replication slot.

Parameter: This function has the same parameters as **pg\_logical\_slot\_peek\_binary\_changes**. For details, see [pg\\_logical\\_slot\\_peek\\_bi...](#)

Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit permissions of the built-in role `gs_role_replication`.

 NOTE

On the CN, this function cannot be executed in a CSN-based replication slot (**confirmed\_csn** is not 0). This function cannot be executed on the standby DN.

- `pg_replication_slot_advance` ('slot\_name', 'upto\_lsn')

Description: Directly updates the streaming replication slot to a specified LSN on the decoding DN, without outputting any decoded result.

Parameters:

- `slot_name`

Indicates the name of the streaming replication slot.

Value range: a string, supporting only lowercase letters, digits, underscores (`_`), question marks (`?`), hyphens (`-`), and periods (`.`). One period or two periods cannot be used as the replication slot name.

- `upto_lsn`

For the CSN-based logical replication slot, it indicates the target CSN before which logs are decoded. During the next decoding, only the transaction results whose CSN is greater than this value will be output. If the input CSN is smaller than the value of **confirmed\_csn** recorded in the current streaming replication slot, the function directly returns the decoded result. If the input CSN is greater than the latest CSN that can be obtained, the latest CSN will be used for decoding.

For the LSN-based logical replication slot, it indicates the target LSN before which logs are decoded. During the next decoding, only the transaction results whose LSN is greater than this value will be output. If an input LSN is smaller than the position recorded in the current streaming replication slot, an error is reported. If the input LSN is greater than the LSN of the current physical log, the latter LSN will be directly used for decoding.

Value range: a string, for example, '1/2AAFC60', '0/A060', or '3A/0' (a hexadecimal uint64 value containing two uint32 values separated by a slash (/); if any uint32 value is 0, 0 is displayed.) **NULL** indicates that the end position of decoding is not specified.

Return type: name, text



**NOTICE**

When a query is performed on a DN, the **confirmed\_csn** query result of the LSN-based logical replication slot is empty, and the **confirmed\_flush** query result of the CSN-based logical replication slot is empty. When a query is performed on the CN, **catalog\_xmin**, **restart\_lsn**, **confirmed\_flush**, and **confirmed\_csn** of the CSN-based logical replication slot are not displayed and the query result is empty.

- `pg_logical_get_area_changes('LSN_start', 'LSN_end', upto_nchanges, 'decoding_plugin', 'xlog_path', 'options_name', 'options_value')`

Description: Specifies an LSN range or an Xlog file for decoding on the decoding DN when no DDL operation is performed.

 **NOTE**

The constraints are as follows:

- The current network and hardware environment are normal.
- It is recommended that the size of a single tuple be less than or equal to 500 MB. If the size ranges from 500 MB to 1 GB, an error is reported.
- Data page replication is not supported for data retrieval that does not fall into Xlogs.
- When an API is called, the log level parameter **wal\_level** must be set to **logical**, and only the log files generated when **wal\_level** is set to **logical** can be parsed. If the used Xlog file is not at the logical level, the decoded content does not have the corresponding value or type. There is no other impact. If **wal\_level** is not set to **logical**, an error is reported and decoding is not performed.
- The Xlog file can be parsed only by a copy of a completely homogeneous DN, and no DDL operation or VACUUM FULL occurs in the database to ensure that the metadata corresponding to the data can be found.
- Do not read too many Xlog files at a time. If no file is specified for decoding within a specified range, you are advised to read one Xlog file each time. Generally, the memory occupied by an Xlog file during decoding is about two to three times the size of the Xlog file.
- Data before VACUUM FULL cannot be retrieved.
- The Xlog file before scale-out cannot be decoded.
- To decode the UPDATE statement, the table must have a primary key. Otherwise, the WHERE clause in the UPDATE statement is empty.
- In this decoding mode, the content that can be decoded is decoded based on the Xlog text record data, and the decoding is not performed based on transactions. Therefore, data that is not in the Xlog cannot be decoded.
- If no decoding file is specified from the decoding point, the system checks whether DDL occurs between the decoding start point and the latest redo value. If DDL occurs, the system does not decode all data. If a decoding file is specified, the system checks whether DDL occurs between the start point of the decoding file and the last readable content of the file and between the start point of the Xlog in the data directory and the latest redo value. If a DDL operation is detected, the system does not decode all tables.

Note: When separation-of-duties is enabled, only the initial database user can call the function. When separation-of-duties is disabled, the SYSADMIN permission is required.

Parameters:

- LSN\_start

Specifies the LSN at the start of decoding.

Value range: a string, in the format of *xlogid/xrecoff*, for example, '1/2AAFC60'. **NULL** indicates that the end position of decoding is not specified.

- LSN\_end

Specifies the LSN at the end of decoding.

Value range: a string, in the format of *xlogid/xrecoff*, for example, '1/2AAFC60'. **NULL** indicates that the end position of decoding is not specified.

- upto\_nchanges

Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto\_nchanges** is set to **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto\_nchanges** after decoding in the first two transactions is finished.

Value range: a non-negative integer

 **NOTE**

If any of the **LSN** and **upto\_nchanges** values are reached, decoding ends.

- decoding\_plugin

Decoding plug-in, which is a .so plug-in that specifies the output format of the decoded content.

Value range: **mppdb\_decoding** and **sql\_decoding**.

- xlog\_path

Decoding plug-in, which specifies the Xlog absolute path and file level of the decoding file.

Value range: **NULL** or a character string of the absolute path of the Xlog file.

- **options**: This parameter is optional and consists of multiple pairs of **options\_name** and **options\_value**. You can retain the default values. For details, see [pg\\_logical\\_slot\\_peek\\_changes](#).

Example:

Perform DML operations on a CN. You can query the DML operations executed on the current DN.  
gaussdb=# execute direct on (datanode1) 'select \* from pg\_logical\_get\_area\_changes("0/502E418", NULL, NULL, "sql\_decoding", NULL);';

```

location | xid | data
-----+-----+-----
0/502E448 | 17365 | insert into public.t1 values (1, 1);
0/502E5A0 | 17365 | COMMIT 17365 (at 2023-11-01 11:28:43.92526+08) 2010016
0/502E5D0 | 17366 | delete from public.t1 where a = 1;insert into public.t1 values (1, 2);
0/502E6D8 | 17366 | COMMIT 17366 (at 2023-11-01 11:28:45.889283+08) 2010017
0/502E7B8 | 17367 | delete from public.t1 where a = 1;
0/502E8B0 | 17367 | COMMIT 17367 (at 2023-11-01 11:28:48.301307+08) 2010018
(6 rows)

```

- **gs\_get\_parallel\_decode\_status()**

Description: Monitors the length of the read log queue and decoding result queue of each decoder thread on the DN where parallel decoding is performed to locate the concurrent decoding performance bottleneck.

Return type: text, int, text, text, text, int64, int64, TimestampTz

## Example:

```
gaussdb=# execute direct on (datanode1) 'select * from gs_get_parallel_decode_status();'
 slot_name | parallel_decode_num | read_change_queue_length | decode_change_queue_length |
 reader_lsn | working_txn_cnt | working_txn_memory | decoded_time
-----+-----+-----+-----+-----+-----+-----+-----
 slot1 | 2 | queue0: 1005, queue1: 320 | queue0: 63, queue1: 748 | 0/1DCE2578
 | 42 | 192927504 | 2023-01-10 11:18:22+08
(1 row)
```

Note: In the return values, **slot\_name** indicates the replication slot name, **parallel\_decode\_num** indicates the number of parallel decoder threads in the replication slot, **read\_change\_queue\_length** indicates the current length of the log queue read by each decoder thread, **decode\_change\_queue\_length** indicates the current length of the decoding result queue of each decoder thread, **reader\_lsn** indicates the log location read by the reader thread, **working\_txn\_cnt** indicates the number of transactions being concatenated in the current sender thread, **working\_txn\_memory** indicates the total memory (in bytes) occupied by the concatenation transactions in the sender thread, and **decoded\_time** indicates the time of the latest WAL decoded by the replication slot.

**NOTICE**

The value of **decoded\_time** comes from checkpoint logs and transaction commit logs, which has a certain error. If no log containing the time is decoded, "2000-01-01 08:00:00+08" (depending on the time zone set in the database) is displayed.

- `gs_get_slot_decoded_wal_time(slot_name)`

Description: Queries the latest decoded WAL time of an active replication slot on the DN where parallel decoding is performed.

Parameters:

- `slot_name`

Specifies the name of the replication slot to be queried.

Value range: a string, supporting only letters, digits, underscores (\_), question marks (?), hyphens (-), and periods (.).

## Example:

```
gaussdb=# execute direct on (datanode1) 'select * from
gs_get_slot_decoded_wal_time("replication_slot");'
gs_get_slot_decoded_wal_time

2023-01-10 11:25:22+08
(1 row)
```

Note: The returned values indicate the time of the latest WAL decoded by the replication slot.

**NOTICE**

The returned time comes from checkpoint logs and transaction commit logs, which has a certain error. If no log containing the time is decoded, "2000-01-01 08:00:00+08" (depending on the time zone set in the database) is displayed. When you query the latest decoded WAL time of a logical replication slot that does not exist, **NULL** is returned. In GSQL, the display of **NULL** is related to the setting, which can be set using `\pset null 'null'`.

- `gs_logical_parallel_decode_status('slot_name')`

Description: Obtains the decoding statistics of an active replication slot for parallel logical decoding on the DN where parallel decoding is performed, including 26 rows of indicators.

The descriptions of the statistical items are listed in the following table.

Record - (stat\_id int, stat\_name TEXT, value TEXT)

**Table 7-65** Statistical items

| Statistical Item             | Description                                                     |
|------------------------------|-----------------------------------------------------------------|
| slot_name                    | Name of the logical replication slot.                           |
| reader_lsn                   | Location of the logic logs to be decoded.                       |
| wal_read_total_time          | Time required for loading the log module.                       |
| wal_wait_total_time          | Time required for waiting for log decoding.                     |
| parser_total_time            | Processing duration of the reader thread.                       |
| decoder_total_time           | Processing duration of all decoder threads.                     |
| sender_total_time            | Processing duration of the sender thread.                       |
| net_send_total_time          | Time required for the network to send logical logs.             |
| net_wait_total_time          | Time required for the network to wait for sending logical logs. |
| net_send_total_bytes         | Number of logical log bytes sent by the network.                |
| transaction_count            | Number of transactions.                                         |
| big_transaction_count        | Number of large transactions.                                   |
| max_transaction_tuples       | Maximum number of transaction operation tuples.                 |
| sent_transaction_count       | Number of transactions sent (by the local database).            |
| spill_disk_transaction_count | Number of flushed transactions.                                 |

| Statistical Item        | Description                                                                                               |
|-------------------------|-----------------------------------------------------------------------------------------------------------|
| spill_disk_bytes        | Total number of bytes flushed to disk.                                                                    |
| spill_disk_count        | Number of disk flushing times.                                                                            |
| input_queue_full_count  | Total number of times that the input queues of all decoder threads are full.                              |
| output_queue_full_count | Total number of times that the output queues of all decoder threads are full.                             |
| dml_count               | Total number of DML statements in WALs decoded by each decoder thread in the local database.              |
| dml_filtered_count      | Total number of DML statements in WALs decoded and filtered by each decoder thread in the local database. |
| toast_count             | Number of modified TOAST table rows.                                                                      |
| candidate_catalog_xmin  | Indicates the <b>catalog_xmin</b> candidate point of the current logical replication slot.                |
| candidate_xmin_lsn      | Updates the log confirmation receiving point required by <b>catalog_xmin</b> .                            |
| candidate_restart_valid | Updates the log confirmation receiving point required by <b>restart_lsn</b> .                             |
| candidate_restart_lsn   | Indicates the <b>restart_lsn</b> candidate point of the current logical replication slot.                 |

Parameters:

- slot\_name

Indicates the name of the streaming replication slot.

Value range: a string, supporting only lowercase letters, digits, underscores (\_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

Return type: int, text, text

Example:

```
gaussdb=# execute direct on (datanode1) 'select * from gs_logical_parallel_decode_status("replication_slot");'
```

| stat_id | stat_name            | value            |
|---------|----------------------|------------------|
| 1       | slot_name            | replication_slot |
| 2       | reader_lsn           | 0/357E180        |
| 3       | wal_read_total_time  | 266694599        |
| 4       | wal_wait_total_time  | 266691307        |
| 5       | parser_total_time    | 39971            |
| 6       | decoder_total_time   | 81216            |
| 7       | sender_total_time    | 48193            |
| 8       | net_send_total_time  | 19388            |
| 9       | net_wait_total_time  | 0                |
| 10      | net_send_total_bytes | 266897           |

```

11 | transaction_count | 7
12 | big_transaction_count | 1
13 | max_transaction_tuples | 4096
14 | sent_transaction_count | 7
15 | spill_disk_transaction_count | 1
16 | spill_disk_bytes | 244653
17 | spill_disk_count | 4096
18 | input_queue_full_count | 0
19 | output_queue_full_count | 0
20 | dml_count | 4097
21 | dml_filtered_count | 0
22 | toast_count | 0
23 | candidate_catalog_xmin | 17152
24 | candidate_xmin_lsn | 0/420A598
25 | candidate_restart_valid | 0/420A598
26 | candidate_restart_lsn | 0/420A598
(26 rows)

```

Note: According to the definitions of statistical items, they must meet the following requirements:

```

wal_read_total_time >= wal_wait_total_time;
transaction_count >= big_transaction_count;
transaction_count >= sent_transaction_count;
transaction_count >= spill_disk_transaction_count;
dml_count >= dml_filtered_count;
dml_count >= toast_count;

```

If `spill_transaction_count == 0`, then `spill_disk_bytes == 0`;

However, frequent locking and unlocking are required, which greatly affects the performance. As a result, the preceding constraints may not be met in extreme cases.

**transaction\_count** indicates the number of transactions in all databases.

**sent\_transaction\_count** indicates the number of transactions sent by the local database because transactions that are not in the local database will not be sent.

If the value of **slot\_name** does not exist, the function does not report an error and the return value is empty.

- `gs_logical_parallel_decode_reset_status('slot_name')`

Description: Resets indicators in `gs_logical_parallel_decode_status('slot_name')` on the DN where parallel decoding is performed.

Parameters:

- `slot_name`

Indicates the name of the streaming replication slot.

Value range: a string, supporting only lowercase letters, digits, underscores (`_`), question marks (`?`), hyphens (`-`), and periods (`.`). One or two periods cannot be used alone as the replication slot name.

Return type: text

Example:

```

gaussdb=# execute direct on (datanode1) 'select * from
gs_logical_parallel_decode_reset_status("replication_slot");
gs_logical_parallel_decode_reset_status

OK

```

```
(1 row)

gaussdb=# execute direct on (datanode1) 'select * from
gs_logical_parallel_decode_status("replication_slot");'
stat_id | stat_name | value
-----+-----+-----
 1 | slot_name | replication_slot
 2 | reader_lsn | 0/357E420
 3 | wal_read_total_time | 0
 4 | wal_wait_total_time | 0
 5 | parser_total_time | 0
 6 | decoder_total_time | 0
 7 | sender_total_time | 0
 8 | net_send_total_time | 0
 9 | net_wait_total_time | 0
10 | net_send_total_bytes | 0
11 | transaction_count | 0
12 | big_transaction_count | 0
13 | max_transaction_tuples | 0
14 | sent_transaction_count | 0
15 | spill_disk_transaction_count | 0
16 | spill_disk_bytes | 0
17 | spill_disk_count | 0
18 | input_queue_full_count | 0
19 | output_queue_full_count | 0
20 | dml_count | 0
21 | dml_filtered_count | 0
22 | toast_count | 0
23 | candidate_catalog_xmin | 0
24 | candidate_xmin_lsn | 0/0
25 | candidate_restart_valid | 0/420A598
26 | candidate_restart_lsn | 0/420A598
(26 rows)
```

Note: If the value of **slot\_name** does not exist, the function does not report an error and the return value is **invalid slot name**.

Do not reset a replication slot that is being observed. The error information is as follows:

- a. If **slot\_name** is empty, the following error is reported: "ERROR: inputString should not be NULL is displayed".
- b. If **slot\_name** is not empty but does not exist, no error is reported but "invalid slot name" is displayed.
- c. If **slot\_name** is not empty but the replication slot corresponding to **slot\_name** is being observed, no error is reported but "can't reset during observing! use gs\_logical\_decode\_stop\_observe to stop." is displayed.

- **gs\_get\_parallel\_decode\_thread\_info()**

Description: Returns the thread information of the active replication slot on the current DN where parallel decoding is performed.

Return type: int64, text, text, int

Example:

```
gaussdb=# execute direct on (decode_datanode1) 'select * from gs_get_parallel_decode_thread_info();'
thread_id | slot_name | thread_type | seq_number
-----+-----+-----+-----
140335364699904 | slot1 | sender | 1
140335214098176 | slot1 | reader | 1
140335325312768 | slot1 | decoder | 1
140335291750144 | slot1 | decoder | 2
140335274968832 | slot1 | decoder | 3
140335258187520 | slot1 | decoder | 4
140335165404928 | slot2 | sender | 1
140335022864128 | slot2 | reader | 1
140335129818880 | slot2 | decoder | 1
```

```
140335113037568 | slot2 | decoder | 2
(10 rows)
```

Note: In the return values, **thread\_id** indicates the thread ID, **slot\_name** indicates the replication slot name, and **thread\_type** indicates the thread type (including the sender, reader and decoder), **seq\_number** indicates the sequence number of each thread with same type in the current replication slot. Each parallel decoding connection only has one sender and reader. Therefore, the sequence numbers of the sender and reader are both 1. The sequence numbers of the decoder are arranged from 1 to the decoding degree of parallelism (DOP) of the current replication slot.

- `gs_get_distribute_decode_status()`

Description: Obtains the distributed decoding status details (by replication slot) on the current node. This command must be executed on CNs. If it is executed on DN, null is returned.

Return type: text, int, int, int64, xid, xid, text, text, text

```
gaussdb=# SELECT * FROM gs_get_distribute_decode_status();
 slot_name | logical_receiver_num | slice_num | walsender_thread_id | last_sent_csn |
last_confirmed_csn | receiver_queue_length | connect_times |
csn_receive_array
-----+-----+-----+-----+-----+-----+-----+-----
 slot1 | 1 | 3 | 139958481843968 | 2012169 | 2010107 | queue0: 1,
queue1: 1, queue2: 0 | slice0: 2, slice1: 2, slice2: 2 | slice0: 2012244, slice1: 2012244, slice2: 2012244
(1 row)
```

**Table 7-66** Statistical items

| Statistical Item      | Type  | Description                                                                                                 |
|-----------------------|-------|-------------------------------------------------------------------------------------------------------------|
| slot_name             | text  | Replication slot name.                                                                                      |
| logical_receiver_num  | int   | Number of receiver threads started for distributed decoding.                                                |
| slice_num             | int   | Number of shards in a cluster.                                                                              |
| walsender_thread_id   | int64 | WAL sender thread ID.                                                                                       |
| last_sent_csn         | xid   | CSN that is sent recently.                                                                                  |
| last_confirmed_csn    | xid   | The last CSN returned from the client that has been confirmed for reception.                                |
| receiver_queue_length | text  | Length of the log queue received on each DN (displayed in the character string format after concatenation). |
| connect_times         | text  | Number of times that each DN is connected (displayed in the character string format after concatenation).   |
| csn_receive_array     | text  | Latest CSN obtained by each DN (displayed in the character string format after concatenation).              |

- `gs_get_distribute_decode_status_detail()`  
Description: Obtains the distributed decoding status details (by DN) on the current node. This command must be executed on CNs. If it is executed on DNs, null is returned.

Return type: text, int, int64, int, int, xid

```
gaussdb=# SELECT * FROM gs_get_distribute_decode_status_detail();
slot_name | slice_id | thread_id | queue_len | connect_times | received_csn
-----+-----+-----+-----+-----+-----
slot1 | 0 | 139959895848704 | 1 | 2 | 2012244
slot1 | 1 | 139959895848704 | 1 | 2 | 2012244
slot1 | 2 | 139959895848704 | 0 | 2 | 2012244
(3 rows)
```

**Table 7-67** Statistical items

| Statistical Item | Type  | Description                                                 |
|------------------|-------|-------------------------------------------------------------|
| slot_name        | text  | Replication slot name.                                      |
| slice_id         | int   | Shard ID (starting from 0).                                 |
| thread_id        | int64 | ID of the receiver thread started for distributed decoding. |
| queue_len        | int   | Length of the log queue received on the current DN.         |
| connect_times    | int   | Number of times that each DN is connected.                  |
| received_csn     | xid   | Latest CSN obtained from the current DN.                    |

### 7.5.25.11 Undo System Functions

 **NOTE**

The Undo system function can be executed only on DNs.

- `gs_undo_meta(type, zoneld, location)`  
Description: Specifies metadata of a module in the undo system.  
Parameter description:
  - **type** (metadata type)
    - **0**: indicates metadata corresponding to an undo zone (record).
    - **1**: indicates metadata corresponding to an undo zone(transaction slot).
    - **2**: indicates metadata corresponding to an undo space (record).
    - **3**: indicates metadata corresponding to an undo space(transaction slot).
  - **zoneld** (undo zone ID)

- -1: indicates metadata of all undo zones.
- 0 to 1024 x 1024 - 1: indicates the meta information of the corresponding zone ID.
- **location** (read location)
  - 0: indicates that data is read from the current memory.
  - 1: indicates that data is read from a physical file.

Return type: record

**Table 7-68** gs\_undo\_meta parameters

| Category         | Parameter   | Type | Description                                                                                                                                        |
|------------------|-------------|------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| Output parameter | zoneld      | oid  | ID of an undo zone.                                                                                                                                |
| Output parameter | persistType | oid  | Persistence level.                                                                                                                                 |
| Output parameter | insert      | text | Position of the next undo record to be inserted.                                                                                                   |
| Output parameter | discard     | text | Position of the undo record that is recycled in common mode.                                                                                       |
| Output parameter | end         | text | Position of the undo record that is forcibly recycled. Values smaller than the value of this parameter indicate that the record has been recycled. |
| Output parameter | used        | text | Undo space that has been used.                                                                                                                     |
| Output parameter | lsn         | text | Modifies the LSN of an undo zone.                                                                                                                  |
| Output parameter | pid         | oid  | ID of a process bound to an undo zone.                                                                                                             |

- gs\_undo\_translot(location, zoneld)  
Description: Specifies transaction slot information of the undo system.  
Parameter description:
  - **location** (read location)

- **0**: indicates that data is read from the current memory.
- **1**: indicates that data is read from a physical file.
- **zoneId** (undo zone ID)
  - **-1**: indicates metadata of all undo zones.
  - **0** to 1024 x 1024 - 1: indicates the meta information of the corresponding zone ID.

Return type: record

**Table 7-69** gs\_undo\_translot parameters

| Category         | Parameter    | Type | Description                                                                                                                                                                                                 |
|------------------|--------------|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Output parameter | groupId      | oid  | Undo zone ID that is used.                                                                                                                                                                                  |
| Output parameter | xactId       | text | Transaction ID.                                                                                                                                                                                             |
| Output parameter | startUndoPtr | text | Position where an undo record is inserted at the start of a transaction corresponding to a transaction slot.                                                                                                |
| Output parameter | endUndoPtr   | text | Position where an undo record is inserted at the end of a transaction corresponding to a transaction slot.                                                                                                  |
| Output parameter | lsn          | text | Transaction slot pointer.                                                                                                                                                                                   |
| Output parameter | slot_states  | oid  | Transaction status. <ul style="list-style-type: none"> <li>• <b>0</b>: committed.</li> <li>• <b>1</b>: being executed.</li> <li>• <b>2</b>: being rolled back.</li> <li>• <b>3</b>: rolled back.</li> </ul> |

- gs\_stat\_undo([bool init])  
Description: Collects undo statistics.  
Return type: record

**Table 7-70** gs\_stat\_undo parameters

| Category         | Parameter              | Type   | Description                                                                                                                                                                                                                                                                                    |
|------------------|------------------------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | init                   | bool   | (Optional) Specifies whether to clear statistics and restart statistics collection.                                                                                                                                                                                                            |
| Output parameter | curr_used_zone_count   | uint32 | Number of used undo zones.                                                                                                                                                                                                                                                                     |
| Output parameter | top_used_zones         | text   | Information about the first three undo zones with the maximum usage. The output format is as follows: <ul style="list-style-type: none"> <li>● <b>zoneId1</b>: used size of zone 1.</li> <li>● <b>zoneId2</b>: used size of zone 2.</li> <li>● <b>zoneId3</b>: used size of zone 3.</li> </ul> |
| Output parameter | curr_used_undo_size    | uint32 | Total size of the undo tablespace that is being used. The unit is MB.                                                                                                                                                                                                                          |
| Output parameter | undo_threshold         | uint32 | Calculation result of the value of the GUC parameter <b>undo_space_limit_size</b> x 80%. The unit is MB.                                                                                                                                                                                       |
| Output parameter | global_recycle_xid     | uint64 | XID of the transaction recycled to the undo space. The undo records generated by the transaction whose XID is smaller than the value of XID are recycled.                                                                                                                                      |
| Output parameter | oldest_xmin            | uint64 | Oldest active transaction.                                                                                                                                                                                                                                                                     |
| Output parameter | total_undo_chain_len   | int64  | Total length of all accessed undo chains.                                                                                                                                                                                                                                                      |
| Output parameter | max_undo_chain_len     | int64  | Maximum length of the accessed undo chain.                                                                                                                                                                                                                                                     |
| Output parameter | create_undo_file_count | uint32 | Number of created undo files.                                                                                                                                                                                                                                                                  |

| Category         | Parameter               | Type   | Description                                                                                                                                                                                                                                                                                                                                       |
|------------------|-------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Output parameter | discard_undo_file_count | uint32 | Number of deleted undo files.                                                                                                                                                                                                                                                                                                                     |
| Output parameter | info                    | text   | If the input parameter is <b>false</b> , suggestions on the <b>undo_space_limit_size</b> , <b>undo_limit_size_per_transaction</b> , and <b>undo_retention_time</b> parameters are provided.<br><br>If the input parameter is <b>true</b> , init (clearing statistics) is required, and only 'The statistics have been initialized.' is displayed. |

Example 1: Clear undo statistics.

```
gaussdb=# select * from gs_stat_undo(true);
curr_used_zone_count | top_used_zones | curr_used_undo_size | undo_threshold | global_recycle_xid
| oldest_xmin | total_undo_chain_len | max_undo_chain_len | create_undo_file_count
| discard_undo_file_count | info
-----+-----+-----+-----+-----
| 3 | 0 : 0, 0 : 0, 0 : 0 | 1 | 209715 | 15741 | 15741
| 0 | 0 |
2 | 0 | The statistics have been initialized.
(1 row)
```

Example 2: Output undo statistics.

```
gaussdb=# select * from gs_stat_undo(false);
curr_used_zone_count | top_used_zones | curr_used_undo_size | undo_threshold | global_recycle_xid
| oldest_xmin | total_undo_chain_len | max_undo_chain_len | create_undo_file_count
| discard_undo_file_count | info
-----+-----+-----+-----+-----
| 3 | 0 : 0, 0 : 0, 0 : 0 | 1 | 209715 | 16253 | 16253
| 0 | 0 |
2 | 0 | Based on the statistic info from last initialization, undo_space_limit_size is
recommended to set >= 120953 blocks, undo_limit_size_per_transaction is
recommended to set <= 327150 blocks, undo_retention_time is recommended to set <= 259200
seconds. Since last initialization, max undo space used by a single transaction is 32715 blocks.
(1 row)
```

- `gs_undo_record(undoptr)`

Description: Parses undo records.

Parameter description: undoptr (undo record pointer)

Return type: record

- `gs_undo_dump_parsepage_mv`(relpath text, blkno bigint, reltype text, rmem boolean)

Description: Parses the page header information of the disk page in the Ustore table, header information of each tuple, flag bit information, and all historical undo version information that can be queried.

Return type: text

Note: Only the system administrator or O&M administrator can execute this function.

 **NOTE**

Currently, this API supports only Ustore tables.

**Table 7-71** `gs_undo_dump_parsepage_mv` parameters

| Category         | Parameter | Type    | Description                                                                                                                                                                                                                                         |
|------------------|-----------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | relpath   | text    | Relative path of the Ustore table data file, in the format of <i>Tablespace name/Database OID/relfilenode</i> , for example, <b>base/16603/16384</b> . You can run the <b>pg_relation_filepath('tablename')</b> command to query the relative path. |
| Input parameter  | blkno     | bigint  | <ul style="list-style-type: none"> <li>• <b>-1</b>: All block pages are parsed.</li> <li>• <b>0–MaxBlocNumber</b>: A specified block page is parsed.</li> </ul>                                                                                     |
| Input parameter  | reltype   | text    | Table type. Currently, only the Ustore table is supported. The value is <b>uheap</b> .                                                                                                                                                              |
| Input parameter  | rmem      | Boolean | <ul style="list-style-type: none"> <li>• <b>false</b></li> <li>• <b>true</b></li> </ul> <p>Currently, the value can only be <b>false</b>, indicating that the corresponding page is parsed from the disk file.</p>                                  |
| Output parameter | output    | text    | Absolute path of the parsing result file.                                                                                                                                                                                                           |

- `gs_undo_meta_dump_zone`(zone\_id int, read\_memory boolean)

Description: Parses undo zone metadata in an undo module.

Return type: record

**Table 7-72** gs\_undo\_meta\_dump\_zone parameters

| Category         | Parameter    | Type    | Description                                                                                                                                                                                      |
|------------------|--------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | zone_id      | int     | Undo zone ID.<br><ul style="list-style-type: none"> <li>• <b>-1</b>: All undo zones are queried.</li> <li>• <b>0-1048575</b>: The undo zone metadata is queried based on the zone ID.</li> </ul> |
| Input parameter  | read_memory  | Boolean | <ul style="list-style-type: none"> <li>• <b>true</b>: Data is read from the current memory.</li> <li>• <b>false</b>: Data is read from the physical file.</li> </ul>                             |
| Output parameter | zone_id      | int     | Undo zone ID.                                                                                                                                                                                    |
| Output parameter | persist_type | int     | Persistence level.<br><ul style="list-style-type: none"> <li>• <b>0</b>: ordinary table</li> <li>• <b>1</b>: unlogged table</li> <li>• <b>2</b>: temporary table</li> </ul>                      |
| Output parameter | insert       | text    | Position of the next undo record to be inserted.                                                                                                                                                 |
| Output parameter | discard      | text    | Position of the undo record that is recycled in common mode.                                                                                                                                     |
| Output parameter | forcediscard | text    | Position of the undo record that is forcibly recycled. Values smaller than the value of this parameter indicate that the record has been recycled.                                               |
| Output parameter | lsn          | text    | Modifies the LSN of a zone.                                                                                                                                                                      |

- gs\_undo\_meta\_dump\_spaces(zone\_id int, read\_memory boolean)  
 Description: Parses metadata of an undo record space and a transaction slot space in an undo module.  
 Return type: record

**Table 7-73** gs\_undo\_meta\_dump\_spaces parameters

| Category         | Parameter             | Type    | Description                                                                                                                                                                                      |
|------------------|-----------------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | zone_id               | int     | Undo zone ID.<br><ul style="list-style-type: none"> <li>• <b>-1</b>: All undo zones are queried.</li> <li>• <b>0-1048575</b>: The undo zone metadata is queried based on the zone ID.</li> </ul> |
| Input parameter  | read_memory           | Boolean | <ul style="list-style-type: none"> <li>• <b>true</b>: Data is read from the current memory.</li> <li>• <b>false</b>: Data is read from the physical file.</li> </ul>                             |
| Output parameter | zone_id               | int     | Undo zone ID.                                                                                                                                                                                    |
| Output parameter | undorecord_space_tail | text    | End position of the undo record space.                                                                                                                                                           |
| Output parameter | undorecord_space_head | text    | Start position of the undo record space.                                                                                                                                                         |
| Output parameter | undorecord_space_lsn  | text    | Modifies the LSN of an undo record space.                                                                                                                                                        |
| Output parameter | undoslot_space_tail   | text    | End position of a transaction slot space.                                                                                                                                                        |
| Output parameter | undoslot_space_head   | text    | Start position of a transaction slot space.                                                                                                                                                      |
| Output parameter | undoreslot_space_lsn  | text    | Modifies the LSN of an undo slot space.                                                                                                                                                          |

- gs\_undo\_meta\_dump\_slot(zone\_id int, read\_memory boolean)  
Description: Parses transaction slot metadata in an undo module.  
Return type: record

**Table 7-74** gs\_undo\_meta\_dump\_slot parameters

| Category         | Parameter          | Type    | Description                                                                                                                                                                                      |
|------------------|--------------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | zone_id            | int     | Undo zone ID.<br><ul style="list-style-type: none"> <li>• <b>-1</b>: All undo zones are queried.</li> <li>• <b>0-1048575</b>: The undo zone metadata is queried based on the zone ID.</li> </ul> |
| Input parameter  | read_memory        | Boolean | <ul style="list-style-type: none"> <li>• <b>true</b>: Data is read from the current memory.</li> <li>• <b>false</b>: Data is read from the physical file.</li> </ul>                             |
| Output parameter | zone_id            | int     | Undo zone ID.                                                                                                                                                                                    |
| Output parameter | allocate           | text    | Allocation position of the undo transaction slot.                                                                                                                                                |
| Output parameter | recycle            | text    | Recycling position of the undo transaction slot.                                                                                                                                                 |
| Output parameter | frozen_xid         | text    | Frozen XID, which is used to determine the visibility.                                                                                                                                           |
| Output parameter | global_frozen_xid  | text    | Minimum frozen XID in the system. Transactions whose XID is smaller than the value of this parameter are visible.                                                                                |
| Output parameter | recycle_xid        | text    | Recycled XID. Transactions whose XID is smaller than the value of this parameter are recycled.                                                                                                   |
| Output parameter | global_recycle_xid | text    | Minimum recycled XID in the system. Transactions whose XID is smaller than the value of this parameter are recycled.                                                                             |

- gs\_undo\_translot\_dump\_slot(zone\_id int, read\_memory boolean)  
Description: Parses a transaction slot in an undo zone.  
Return type: record

**Table 7-75** gs\_undo\_translot\_dump\_slot parameters

| Category         | Parameter      | Type    | Description                                                                                                                                                                                              |
|------------------|----------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | zone_id        | oid     | Undo zone ID.<br><ul style="list-style-type: none"> <li>• <b>-1</b>: All undo zones are queried.</li> <li>• <b>0-1048575</b>: The undo zone metadata is queried based on the zone ID.</li> </ul>         |
| Input parameter  | read_memory    | Boolean | <ul style="list-style-type: none"> <li>• <b>true</b>: Data is read from the current memory.</li> <li>• <b>false</b>: Data is read from the physical file.</li> </ul>                                     |
| Output parameter | zone_id        | text    | Undo zone ID.                                                                                                                                                                                            |
| Output parameter | slot_xid       | text    | Transaction ID.                                                                                                                                                                                          |
| Output parameter | start_undo_ptr | text    | Position where an undo record is inserted at the start of a transaction corresponding to a transaction slot.                                                                                             |
| Output parameter | end_undo_ptr   | text    | Position where an undo record is inserted at the end of a transaction corresponding to a transaction slot.                                                                                               |
| Output parameter | slot_ptr       | text    | Position of a transaction slot.                                                                                                                                                                          |
| Output parameter | slot_states    | oid     | Transaction state<br><ul style="list-style-type: none"> <li>• <b>0</b>: committed</li> <li>• <b>1</b>: being executed</li> <li>• <b>2</b>: being rolled back</li> <li>• <b>3</b>: rolled back</li> </ul> |

- gs\_undo\_translot\_dump\_xid(slot\_xid xid, read\_memory boolean)  
Description: Parses a transaction slot in an undo zone based on the XID.  
Return type: record

**Table 7-76** gs\_undo\_translot\_dump\_xid parameters

| Category         | Parameter      | Type    | Description                                                                                                                                                                                           |
|------------------|----------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | slot_xid       | xid     | Transaction ID to be queried.                                                                                                                                                                         |
| Input parameter  | read_memory    | Boolean | <ul style="list-style-type: none"> <li>• <b>true</b>: Data is read from the current memory.</li> <li>• <b>false</b>: Data is read from the physical file.</li> </ul>                                  |
| Output parameter | zone_id        | text    | Undo zone ID.                                                                                                                                                                                         |
| Output parameter | slot_xid       | text    | Transaction ID.                                                                                                                                                                                       |
| Output parameter | start_undo_ptr | text    | Position where an undo record is inserted at the start of a transaction corresponding to a transaction slot.                                                                                          |
| Output parameter | end_undoptr    | text    | Position where an undo record is inserted at the end of a transaction corresponding to a transaction slot.                                                                                            |
| Output parameter | slot_ptr       | text    | Position of a transaction slot.                                                                                                                                                                       |
| Output parameter | slot_states    | oid     | Transaction state <ul style="list-style-type: none"> <li>• <b>0</b>: committed</li> <li>• <b>1</b>: being executed</li> <li>• <b>2</b>: being rolled back</li> <li>• <b>3</b>: rolled back</li> </ul> |

- gs\_undo\_dump\_record(undoptr bigint)  
Description: Parses undo records.  
Return type: record

**Table 7-77** gs\_undo\_dump\_record parameters

| Category        | Parameter | Type   | Description                                     |
|-----------------|-----------|--------|-------------------------------------------------|
| Input parameter | undoptr   | bigint | Start position of the undo record to be parsed. |

| Category         | Parameter    | Type   | Description                                             |
|------------------|--------------|--------|---------------------------------------------------------|
| Output parameter | undoptr      | bigint | Start position of the undo record to be parsed.         |
| Output parameter | xactid       | text   | Transaction ID.                                         |
| Output parameter | cid          | text   | Command ID.                                             |
| Output parameter | reloid       | text   | Relation OID.                                           |
| Output parameter | relfilenode  | text   | Relfinode of the file.                                  |
| Output parameter | utype        | text   | Undo record type.                                       |
| Output parameter | blkprev      | text   | Position of the previous undo record in the same block. |
| Output parameter | blockno      | text   | Block number.                                           |
| Output parameter | uoffset      | text   | Undo record offset.                                     |
| Output parameter | prevurp      | text   | Position of the previous undo record.                   |
| Output parameter | payloadlen   | text   | Length of the undo record data.                         |
| Output parameter | oldxactid    | text   | Previous transaction ID.                                |
| Output parameter | partitionoid | text   | Partition OID.                                          |



**Table 7-78** gs\_undo\_dump\_xid parameters

| Category         | Parameter   | Type | Description                                             |
|------------------|-------------|------|---------------------------------------------------------|
| Input parameter  | undo_xid    | xid  | Transaction ID.                                         |
| Output parameter | undoptr     | xid  | Start position of the undo record to be parsed.         |
| Output parameter | xactid      | text | Transaction ID.                                         |
| Output parameter | cid         | text | Command ID.                                             |
| Output parameter | reloid      | text | Relation OID.                                           |
| Output parameter | relfilenode | text | Relfinode of the file.                                  |
| Output parameter | utype       | text | Undo record type.                                       |
| Output parameter | blkprev     | text | Position of the previous undo record in the same block. |
| Output parameter | blockno     | text | Block number.                                           |
| Output parameter | uoffset     | text | Undo record offset.                                     |
| Output parameter | prevurp     | text | Position of the previous undo record.                   |
| Output parameter | payloadlen  | text | Length of the undo record data.                         |
| Output parameter | oldxactid   | text | Previous transaction ID.                                |

| Category         | Parameter         | Type | Description                                              |
|------------------|-------------------|------|----------------------------------------------------------|
| Output parameter | partitionoid      | text | Partition OID.                                           |
| Output parameter | tablespace        | text | Tablespace.                                              |
| Output parameter | alreadyread_bytes | text | Length of the read undo record.                          |
| Output parameter | prev_undo_rec_len | text | Length of the previous undo record.                      |
| Output parameter | td_id             | text | ID of the transaction directory.                         |
| Output parameter | reserved          | text | Reserved flag of the old tuple stored in an undo record. |
| Output parameter | flag              | text | Status flag of the old tuple stored in an undo record.   |
| Output parameter | flag2             | text | Number of old tuple columns stored in an undo record.    |
| Output parameter | t_hoff            | text | Length of the undo record data header.                   |

- `gs_verify_undo_record(type, start_idx, end_idx, location)`

Description: Verifies undo records. Currently, only the disk verification mode is supported. Offline verification can be performed only when services are not running. Before the verification, you need to manually perform checkpoint flushing.

Return type: record

**Table 7-79** gs\_verify\_undo\_record parameters

| Category         | Parameter | Type  | Description                                                                                                                                                                                                                                             |
|------------------|-----------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | type      | text  | Verification type. <ul style="list-style-type: none"> <li>'urp': verifies all undo records in a specified URP range.</li> <li>'zone': verifies all undo records of all zones in a specified zone range.</li> </ul>                                      |
| Input parameter  | start_idx | int64 | Start position. <ul style="list-style-type: none"> <li>When <b>type</b> is set to 'urp', this parameter indicates the start position of undo records.</li> <li>When type is set to 'zone', this parameter indicates the start undo zone ID.</li> </ul>  |
| Input parameter  | end_idx   | int64 | End position. <ul style="list-style-type: none"> <li>When <b>type</b> is set to 'urp', this parameter indicates the end position of undo records.</li> <li>When <b>type</b> is set to 'zone', this parameter indicates the end undo zone ID.</li> </ul> |
| Input parameter  | location  | bool  | <ul style="list-style-type: none"> <li>0: memory verification</li> <li>1: disk verification</li> </ul> Currently, this parameter can only be set to 1.                                                                                                  |
| Output parameter | zone_id   | int64 | Undo zone ID.                                                                                                                                                                                                                                           |
| Output parameter | detail    | text  | Verification error information.                                                                                                                                                                                                                         |

Example 1: Verify the undo record whose URP is 24.

```
gaussdb=# select * from gs_verify_undo_record('urp', 24, 24, 1);
zone_id | detail
-----+-----
(0 rows)
```

Example 2: If the entered value of **urp** is incorrect, that is, the value is not the start position of the undo record, the system may return a message indicating that the verification result is empty or the verification fails.

```
gaussdb=# select * from gs_verify_undo_record('urp', 35184372173095, 35184372173095, 1);
zone_id | detail
-----+-----
```

```
2 | verification failed: xid is invalid, urp:35184372173095.
(1 row)
```

Example 3: Verify all undo records from zone 0 to zone 2 on the disk.

```
gaussdb=# select * from gs_verify_undo_record('zone', 0, 2, 1);
zone_id | detail
-----+-----
(0 rows)
```

 **NOTE**

If an error is reported when this view is called, contact Huawei technical support.

- `gs_verify_undo_slot(type, start_idx, end_idx, location)`  
Description: Verifies undo transaction slots. Currently, only the disk verification mode is supported. Offline verification can be performed only when services are not running. Before the verification, you need to manually perform checkpoint flushing.  
Return type: record

**Table 7-80** `gs_verify_undo_slot` parameters

| Category         | Parameter | Type  | Description                                                                                                                                                                      |
|------------------|-----------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | type      | text  | Verification type. <ul style="list-style-type: none"> <li>• <b>'zone'</b>: verifies all transaction slots of all zones in a specified zone range.</li> </ul>                     |
| Input parameter  | start_idx | int64 | Start undo zone ID.                                                                                                                                                              |
| Input parameter  | end_idx   | int64 | End undo zone ID.                                                                                                                                                                |
| Input parameter  | location  | bool  | <ul style="list-style-type: none"> <li>• <b>0</b>: memory verification</li> <li>• <b>1</b>: disk verification</li> </ul> Currently, this parameter can only be set to <b>1</b> . |
| Output parameter | zone_id   | int64 | Undo zone ID.                                                                                                                                                                    |
| Output parameter | detail    | text  | Verification error information.                                                                                                                                                  |

Example 1: Verify all transaction slot records from zone 0 to zone 2 on the disk.

```
gaussdb=# select * from gs_verify_undo_slot('zone', 0, 2, 1);
zone_id | detail
```

```
-----+-----
(0 rows)
```

 **NOTE**

If an error is reported when this view is called, contact Huawei technical support.

- `gs_verify_undo_meta(type, start_idx, end_idx, location)`  
Description: Verifies undo metadata. Currently, only the disk verification mode is supported. Offline verification can be performed only when services are not running. Before the verification, you need to manually perform checkpoint flushing.  
Return type: record

**Table 7-81** `gs_verify_undo_meta` parameters

| Category         | Parameter | Type  | Description                                                                                                                                                                                            |
|------------------|-----------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | type      | text  | Verification type. The value of <b>type</b> can only be set to 'all'. <ul style="list-style-type: none"> <li>• 'all': verifies all meta information of all zones in a specified zone range.</li> </ul> |
| Input parameter  | start_idx | int64 | Start undo zone ID.                                                                                                                                                                                    |
| Input parameter  | end_idx   | int64 | End undo zone ID.                                                                                                                                                                                      |
| Input parameter  | location  | bool  | <ul style="list-style-type: none"> <li>• 0: memory verification</li> <li>• 1: disk verification</li> </ul> Currently, this parameter can only be set to 1.                                             |
| Output parameter | zone_id   | int64 | Undo zone ID.                                                                                                                                                                                          |
| Output parameter | detail    | text  | Verification error information.                                                                                                                                                                        |

Example 1: Verify all meta information records from zone 0 to zone 2 on the disk.

```
gaussdb=# select * from gs_verify_undo_meta('all', 0, 2, 1);
zone_id | detail
-----+-----
(0 rows)
```

 **NOTE**

If an error is reported when this view is called, contact Huawei technical support.

- `gs_async_rollback_worker_status()`  
Description: Monitors the status of active asynchronous rollback threads.

Return type: record

**Table 7-82** gs\_async\_rollback\_worker\_status parameters

| Category         | Parameter           | Type       | Description                                                                                                                           |
|------------------|---------------------|------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Output parameter | datid               | oid        | Database ID.                                                                                                                          |
| Output parameter | pid                 | int64      | Process ID.                                                                                                                           |
| Output parameter | sessionid           | int64      | Session ID.                                                                                                                           |
| Output parameter | usesysid            | oid        | ID of the user who initiates the thread.                                                                                              |
| Output parameter | state               | int32      | Thread status.<br><b>0</b> : undefined<br><b>1</b> : idle<br><b>2</b> : running                                                       |
| Output parameter | rollback_start_time | timestampz | Timestamp when a thread is started.                                                                                                   |
| Output parameter | idx                 | uint32     | Index of an asynchronous rollback thread in the array.                                                                                |
| Output parameter | xid                 | uint64     | XID of the transaction that is being rolled back.                                                                                     |
| Output parameter | progress            | text       | Rollback progress of the transaction (number of undo records that have been rolled back/total number of undo records, in percentage). |

- gs\_async\_rollback\_xact\_status()  
Description: Monitors the hash table of asynchronous rollback tasks.  
Return type: record

**Table 7-83** gs\_async\_rollback\_xact\_status parameters

| Category         | Parameter     | Type   | Description                                                       |
|------------------|---------------|--------|-------------------------------------------------------------------|
| Output parameter | xid           | xid    | XID of the transaction that requires asynchronous rollback.       |
| Output parameter | start_undoptr | uint64 | Pointer to the start undo record of the transaction.              |
| Output parameter | end_undoptr   | uint64 | Pointer to the end undo record of the transaction.                |
| Output parameter | dbid          | uint32 | ID of the database where the transaction is located.              |
| Output parameter | slot_ptr      | uint64 | Pointer to the transaction slot corresponding to the transaction. |
| Output parameter | launched      | bool   | Checks whether there is an active asynchronous rollback thread.   |

- gs\_undo\_recycler\_status()  
Description: Monitors the status of asynchronous recycling threads.  
Return type: record

**Table 7-84** gs\_undo\_recycler\_status parameters

| Category         | Parameter     | Type       | Description                                                                     |
|------------------|---------------|------------|---------------------------------------------------------------------------------|
| Output parameter | datid         | oid        | Database ID.                                                                    |
| Output parameter | pid           | int64      | Process ID.                                                                     |
| Output parameter | sessionid     | int64      | Session ID.                                                                     |
| Output parameter | usesysid      | oid        | ID of the user who initiates the thread.                                        |
| Output parameter | state         | int32      | Thread status.<br><b>0</b> : undefined<br><b>1</b> : idle<br><b>2</b> : running |
| Output parameter | backend_start | timestampz | Timestamp when a thread is started.                                             |

| Category         | Parameter                                | Type   | Description                                         |
|------------------|------------------------------------------|--------|-----------------------------------------------------|
| Output parameter | total_recycle_time                       | uint64 | Total recycling time.                               |
| Output parameter | max_recycle_time                         | uint64 | Maximum recycling time.                             |
| Output parameter | total_recycle_size                       | uint64 | Total recycled space.                               |
| Output parameter | total_recycle_count                      | uint64 | Total number of recycling times.                    |
| Output parameter | recycle_sleep_count                      | uint64 | Number of sleep times.                              |
| Output parameter | recycle_sleep_time                       | uint64 | Total sleep time.                                   |
| Output parameter | max_recycle_sleep_time                   | uint64 | Maximum sleep time.                                 |
| Output parameter | last_recycle_timestamp                   | uint64 | Timestamp of the last successful recycling.         |
| Output parameter | last_update_global_recycle_xid_timestamp | uint64 | Timestamp of the last global recycling transaction. |

- `gs_undo_launcher_status()`  
Description: Monitors the status of asynchronous rollback launcher threads.  
Return type: record

**Table 7-85** `gs_undo_launcher_status` parameters

| Category         | Parameter | Type  | Description                                                                     |
|------------------|-----------|-------|---------------------------------------------------------------------------------|
| Output parameter | datid     | oid   | Database ID.                                                                    |
| Output parameter | pid       | int64 | Process ID.                                                                     |
| Output parameter | sessionid | int64 | Session ID.                                                                     |
| Output parameter | usesysid  | oid   | ID of the user who initiates the thread.                                        |
| Output parameter | state     | int32 | Thread status.<br><b>0</b> : undefined<br><b>1</b> : idle<br><b>2</b> : running |

| Category         | Parameter                               | Type        | Description                                                                                            |
|------------------|-----------------------------------------|-------------|--------------------------------------------------------------------------------------------------------|
| Output parameter | backend_start                           | timestamptz | Timestamp when a thread is started.                                                                    |
| Output parameter | total_async_roll<br>back_task_coun<br>t | uint64      | Total number of asynchronous rollback tasks initiated after the database on the local node is started. |
| Output parameter | average_async_<br>rollback_time         | uint64      | Average duration of an asynchronous rollback task.                                                     |
| Output parameter | max_async_roll<br>back_time             | uint64      | Maximum duration of an asynchronous rollback task.                                                     |
| Output parameter | min_async_rollb<br>ack_time             | uint64      | Minimum duration of an asynchronous rollback task.                                                     |

### 7.5.25.12 Other Functions

- `pgxc_pool_check()`  
Description: Checks whether the connection data buffered in the pool is consistent with **pgxc\_node**.  
Return type: Boolean
- `pgxc_pool_reload()`  
Description: Updates the connection information buffered in the pool.  
Return type: Boolean
- `reload_active_coordinator()`  
Description: Updates the connection information buffered in the pool for all active CNs.  
Return type: void
- `pgxc_lock_for_backup()`  
Description: Locks a cluster for taking backup that would be restored on the new node to be added.  
Return type: Boolean

#### NOTE

**pgxc\_lock\_for\_backup** locks a cluster before **gs\_dump** or **gs\_dumpall** is used to back up the cluster. After a cluster is locked, operations changing the system structure are not allowed. This function does not affect DML statements.

- `pg_pool_validate(clear bool, node_name text)`  
Description: Displays invalid connections in the pooler between the CN and node\_name. When the value of **clear** is **true**, invalid connections are cleared.  
Return type: record

- `gs_validate_node_conn(validate_type cstring, validate_node_name cstring)`  
Description: Displays or clears the connections received by the current node from a specified node.  
Return type: record

**Table 7-86** `gs_validate_node_conn` parameters

| Category         | Parameter                       | Type                 | Description                                                                                                                                                                                                                                                                               |
|------------------|---------------------------------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | <code>validate_type</code>      | <code>cstring</code> | Specifies an input parameter type. <ul style="list-style-type: none"> <li>• <b>'check'</b>: displays the connections received by the current node from a specified node.</li> <li>• <b>'clear'</b>: clears the connections received by the current node from a specified node.</li> </ul> |
| Input parameter  | <code>validate_node_name</code> | <code>cstring</code> | Specific a node name.                                                                                                                                                                                                                                                                     |
| Output parameter | <code>pid</code>                | <code>bigint</code>  | Displays the ID of a thread where the current node receives connections from the specified node. If the thread is from a thread pool, the value is <b>0</b> .                                                                                                                             |
| Output parameter | <code>sessionid</code>          | <code>bigint</code>  | Displays the ID of a session where the current node receives connections from the specified node. If the thread is not from a thread pool, the value is <b>0</b> .                                                                                                                        |
| Output parameter | <code>node_name</code>          | <code>text</code>    | Displays the name of a node from which the current node receives connections.                                                                                                                                                                                                             |

- `pgxc_pool_connection_status()`  
Description: Checks whether the pooler connection status is **normal**.  
Return type: Boolean
- `pg_nodes_memory()`  
Description: Queries the memory usage of all nodes.  
Return type: record
- `table_skewness(text)`  
Description: Queries the percentage of table data among all nodes.  
Parameter: Indicates that the type of the name of the to-be-queried table is text.  
Return type: record

- table\_skewness(text, text, text)**

Description: Queries the percentage of a specified column in the table data among all nodes.

Parameters: name of the table to be queried, specified column name, and number of records in the specified table. The default value is **0**, indicating that all records are queried. All parameters are of the text type.

Return type: record

Return value description: Node ID, number of data rows in a specified column, and percentage of the data volume of the current node to the total data volume

Example:

Return the distribution of the first five rows of data in the **a** column of the **t** table on a node.

```
gaussdb=# select table_skewness('t', 'a',5);
table_skewness

(1,3,60.000%)
(2,2,40.000%)
(2 rows)
```

Return the distribution of all data in the **a** column of the **t** table on a node.

```
gaussdb=# select table_skewness('t', 'a');
table_skewness

(1,7,70.000%)
(2,2,20.000%)
(0,1,10.000%)
(3 rows)
```
- table\_skewness\_with\_schema(text, text)**

Description: Checks the proportion of table data on all nodes. The function is the same as that of **table\_skewness(text)**.

**text** indicates that the types of the schema name and table name for the table to be queried are both text.

Return type: record
- table\_data\_skewness(colrecord, type)**

Description: Queries the node where the table data is located.

Parameter description:

**colrecord**: column name record of the table to be queried. The value is of the record type.

**type**: hash distribution type

Return type: smallint

Example:

```
-- Return the node where the index column data of the 'test1' table is located.
gaussdb=# select table_data_skewness(row(index), 'R') from test1;
table_data_skewness

4
3
1
2
(4 rows)
```
- table\_distribution(schemaname text, tablename text)**

Description: Queries the storage space occupied by a specified table on each node.

**text** indicates that the types of the schema name and table name for the table to be queried are both text.

Return type: record

 **NOTE**

- To query the storage distribution of a specified table by using this function, you must have the SELECT permission for the table.
- The performance of **table\_distribution** is better than that of **table\_skewness**. In the scenario with a large amount of data, **table\_distribution** is recommended.
- When you use **table\_distribution** and want to view the space usage, you can use **dnsize** or **(sum(dnsize) over ())** to view the percentage.

- **table\_distribution()**

Description: Queries the storage distribution of all tables in the current database.

Return type: record

 **NOTE**

- This function involves the query for information about all tables in the database. To execute this function, you must have the administrator rights.
- Based on the **table\_distribution()** function, GaussDB provides the **PGXC\_GET\_TABLE\_SKEWNESS** view as an alternative way to query data skew. You are advised to use this view when the number of tables in the database is less than 10000.

- **plan\_seed()**

Description: Obtains the seed value of the previous query statement (internal use).

Return type: int

- **pg\_stat\_get\_env()**

Description: Obtains the environment variable information of the current node. Only users with the SYSADMIN or MONADMIN permission can access the environment variable information.

Return type: record

Example:

```
gaussdb=# select pg_stat_get_env();
 pg_stat_get_env

(coordinator1,localhost,144773,49100,/data1/GaussDB_Kernel_TRUNK/install,/data1/
GaussDB_Kernel_TRUNK/install/data/coordinator1,pg_log)
(1 row)
```

- **pg\_catalog.plancache\_clean()**

Description: Clears the global plan cache that is not used on nodes.

Return type: Boolean

- **pg\_stat\_get\_thread()**

Description: Provides thread status information on the current node. Users with the SYSADMIN or MONADMIN permission can view information about all threads. Common users can view only their own thread information.

Return type: record

- `pgxc_get_os_threads()`  
Description: Provides thread status information about all normal nodes in the entire cluster.  
Return type: record
- `pg_stat_get_sql_count()`  
Description: Provides the counts of the SELECT, UPDATE, INSERT, DELETE, and MERGE INTO statements executed on the current node. Users with the SYSADMIN or MONADMIN permission can view information about all users. Common users can view only their own statistics.  
Return type: record
- `pgxc_get_sql_count()`  
Description: Provides the counts of the SELECT, UPDATE, INSERT, DELETE, and MERGE INTO statements executed on all the nodes in the entire cluster.  
Return type: record
- `pgxc_get_node_env()`  
Description: Provides the environment variable information about all nodes in a cluster.  
Return type: record
- `pgxc_disaster_read_set(text)`  
Description: Configures node information about the standby cluster for DR on ETCD. Only the standby cluster for DR is available and can be called only by the initial user.  
Return type: Boolean
- `pgxc_disaster_read_init()`  
Description: Initializes readable DR resources and status information. Only the standby cluster for DR is available and can be called only by the initial user.  
Return type: Boolean
- `pgxc_disaster_read_clear()`  
Description: Clears readable DR resources and status information. Only the standby cluster for DR is available and can be called only by the initial user.  
Return type: Boolean
- `pgxc_disaster_read_status()`  
Description: Provides node information about the standby cluster for DR. This function is available only for the standby cluster for DR.  
Return type: record
- `gs_switch_relfilenode()`  
Description: Exchanges meta information of two tables or partitions. (This is only used for the redistribution tool. An error message is displayed when the function is directly used by users).  
Return type: int

 **NOTE**

This function may cause clearing of statistics. You are advised to collect statistics again after this function is called.

- `pg_catalog.plancache_clean()`  
Description: Clears the global plan cache that is not used on the current node.  
Return type: Boolean
- `DBE_PERF.global_plancache_clean()`  
Description: Clears the global plan cache that is not used on all nodes.  
Return type: Boolean
- `copy_error_log_create()`  
Description: Creates the error table (**public.pgxc\_copy\_error\_log**) required for creating the **COPY FROM** error tolerance mechanism.  
Return type: Boolean

 NOTE

- This function attempts to create the **public.pgxc\_copy\_error\_log** table. For details about the table, see [Table 7-87](#).
- In addition, it creates a B-tree index on the **relname** column and executes **REVOKE ALL on public.pgxc\_copy\_error\_log FROM public** to manage permissions on the error table (the permissions are the same as those of the **COPY** statement).
- **public.pgxc\_copy\_error\_log** is a row-store table. Therefore, this function can be executed and **COPY** error tolerance is available only when row-store tables can be created in the cluster. Row-store tables cannot be created in the cluster if the GUC parameter **enable\_hadoop\_env** is set to **on** (by default, this parameter set to **off** for GaussDB).
- Same as the error table and the **COPY** statement, the function requires **SYSADMIN** or higher permissions.
- If the **public.pgxc\_copy\_error\_log** table or the **copy\_error\_log\_relname\_idx** index exists before the function creates it, the function will report an error and roll back.

**Table 7-87** Error table `public.pgxc_copy_error_log`

| Column                 | Type                     | Description                                                              |
|------------------------|--------------------------|--------------------------------------------------------------------------|
| <code>relname</code>   | character varying        | Table name in the form of <i>Schema name.Table name</i>                  |
| <code>begintime</code> | timestamp with time zone | Time when a data format error was reported                               |
| <code>filename</code>  | character varying        | Name of the source data file where a data format error occurs            |
| <code>lineno</code>    | bigint                   | Number of the row where a data format error occurs in a source data file |
| <code>rawrecord</code> | text                     | Raw record of a data format error in the source data file                |
| <b>detail</b>          | text                     | Error details                                                            |

- `pg_stat_get_data_senders()`  
Description: Provides detailed information about the data-copy sender thread active at the moment.

- Return type: record
- `textlen()`  
Description: Provides the method of querying the logical length of text.  
Return type: int
- `threadpool_status()`  
Description: Displays the status of worker threads and sessions in the thread pool.  
Return type: record
- `get_local_active_session()`  
Description: Provides sampling records of historical active sessions stored in the memory by current node. Users with the SYSADMIN or MONADMIN permission can view all historical active session records of the current node. Common users can view the historical active session records of the current session.  
Return type: record
- `dbe_perf.get_global_active_session()`  
Description: Provides sampling records of the historical active sessions stored in the memory of all nodes.  
Return type: record
- `dbe_perf.get_global_gs_asp(timestamp, timestamp)`  
Description: Provides sampling records of the historical active sessions stored in the `gs_asp` system catalog of all nodes.  
Return type: record
- `get_wait_event_info()`  
Description: Provides detailed information about the wait event.  
Return type: record
- `dbe_perf.get_datanode_active_session(text)`  
Description: Provides sampling records of historical active sessions stored in the memory of DN, which is queried from CN.  
Return type: record  
Note: This function queries the records in the `local_active_session` view on the target DN and matches the records with those in the `local_active_session` view on all CNs to obtain the query string. Therefore, a large amount of memory is occupied.
- `dbe_perf.get_datanode_active_session_hist(text, timestamp, timestamp)`  
Description: Provides sampling records of historical active sessions stored in the `gs_asp` system catalog of DN, which is queried from CN.  
Return type: record  
Note: This function queries the **gs\_asp** records of a specified period on the target DN. If the period is specified too long, too many records will be queried, which takes a long time.
- `generate_wdr_report(bigint, bigint, cstring, cstring, cstring)`  
Description: Generates a system diagnosis report based on two snapshots. By default, the initial user or MONADMIN can access the report. The result can

be queried only in the system database but cannot be queried in the user database.

Return type: text

**Table 7-88** generate\_wdr\_report parameter description

| Parameter     | Description                                                                                                                                                                                                                                                                    | Range                                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| begin_snap_id | Snapshot ID that starts the diagnosis report period.                                                                                                                                                                                                                           | N/A                                                                                                                                                                          |
| end_snap_id   | Snapshot ID that ends the diagnosis report period. By default, the value of <b>end_snap_id</b> is greater than that of <b>begin_snap_id</b> .                                                                                                                                  | N/A                                                                                                                                                                          |
| report_type   | Specifies the type of the generated report.                                                                                                                                                                                                                                    | <ul style="list-style-type: none"> <li>• <b>summary</b></li> <li>• <b>detail</b></li> <li>• <b>all</b>: Both <b>summary</b> and <b>detail</b> types are included.</li> </ul> |
| report_scope  | Specifies the scope for a report to be generated.                                                                                                                                                                                                                              | <ul style="list-style-type: none"> <li>• <b>cluster</b>: database-level information</li> <li>• <b>node</b>: node-level information.</li> </ul>                               |
| node_name     | <ul style="list-style-type: none"> <li>• When <b>report_scope</b> is set to <b>node</b>, set this parameter to the name of the corresponding node.</li> <li>• If <b>report_scope</b> is set to <b>cluster</b>, this parameter can be omitted or set to <b>NULL</b>.</li> </ul> | <ul style="list-style-type: none"> <li>• <b>node</b>: node name in GaussDB.</li> <li>• <b>cluster</b>: This value is omitted, left blank or set to <b>NULL</b>.</li> </ul>   |

- create\_wdr\_snapshot()

Description: Manually generates system diagnosis snapshots. This function requires the SYSADMIN permission and can be executed only on the CCN.

Return type: text

- kill\_snapshot()

Description: Kills the WDR snapshot backend thread. Users who call this function must have the SYSADMIN permission, the REPLICATION permission, or inherit permissions of the built-in role gs\_role\_replication.

Return type: void

- capture\_view\_to\_json(text, integer)

Description: Saves the view result to the directory specified by GUC: **perf\_directory**. If **is\_crossdb** is set to **1**, the view is accessed once for all

databases. If the value of **is\_crossdb** is **0**, the current database is accessed only once. Only users with the SYSADMIN or MONADMIN permission can execute this function.

Return type: int

- `reset_unique_sql(text, text, bigint)`

Description: Clears the Unique SQL statements in the memory of CN/DN. (The SYSADMIN or MONADMIN permission is required.)

Return type: Boolean

**Table 7-89** reset\_unique\_sql parameter description

| Parameter   | Type | Description                                                                                                                                                                                                             |
|-------------|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| scope       | text | Clearance scope type. The options are as follows:<br><b>'GLOBAL'</b> : Clears all CNs/DNs. If the value is <b>'GLOBAL'</b> , this function can be executed only on the CN.<br><b>'LOCAL'</b> : Clears the current node. |
| clean_type  | text | <b>'BY_USERID'</b> : Unique SQL statements are cleared based on user IDs.<br><b>'BY_CNID'</b> : Unique SQL statements are cleared based on CN IDs.<br><b>'ALL'</b> : All data is cleared.                               |
| clean_value | int8 | Clearance value corresponding to the clearance type. If the second parameter is set to <b>ALL</b> , the third parameter does not take effect and can be set to any value.                                               |

- `wdr_xdb_query(db_name_str text, query text)`

Description: Provides the capability of executing local cross-database queries. For example, when connecting to the **testdb** database, access tables in the **test** database. Only the initial user has the permission to run this command.

```
select col1 from wdr_xdb_query('dbname=test','select col1 from t1') as dd(col1 int);
```

Return type: record

- `pg_wlm_jump_queue(pid int)`

Description: Moves a task to the top of the queue of CN.

Return type: Boolean

- **true**: success
- **false**: failure

- `gs_wlm_switch_cgroup(pid int, cgroup text)`

Description: Moves a job to another Cgroup to change the job priority.

Return type: Boolean

- **true**: success

- **false**: failure
- `pv_session_memctx_detail(threadid tid, MemoryContextName text)`  
Description: Records information about the memory context **MemoryContextName** of the thread **tid** into the `threadid_timestamp.log` file in the `$GAUSSLOG/pg_log/${node_name}/dumpmem` directory. `threadid` can be obtained from `sessid` in the **PV\_SESSION\_MEMORY\_DETAIL** table. In the officially released version, only the **MemoryContextName** that is an empty string (two single quotation marks indicate that the input is an empty string) is accepted. In this case, all memory context information is recorded. Otherwise, no operation is performed. This function can be executed only by the administrator.  
Return type: Boolean
  - **true**: success
  - **false**: failure
- `pg_shared_memctx_detail(MemoryContextName text)`  
Description: Records information about the memory context **MemoryContextName** into the `threadid_timestamp.log` file in the `$GAUSSLOG/pg_log/${node_name}/dumpmem` directory. Calling this function in the officially released version does not involve any operation. Only the administrator can execute this function.  
Return type: Boolean
  - **true**: success
  - **false**: failure
- `local_bgwriter_stat()`  
Description: Displays the information about pages flushed by the bgwriter thread of this instance, number of pages in the candidate buffer chain, and buffer elimination information.  
Return type: record
- `local_candidate_stat()`  
Description: Displays the number of pages in the candidate buffer chain of this instance and buffer elimination information, including the normal buffer pool and segment buffer pool.  
Return type: record
- `local_ckpt_stat()`  
Description: Displays the information about checkpoints and flushing pages of the current instance.  
Return type: record
- `local_double_write_stat()`  
Description: Displays the doublewrite file status of the current instance.  
Return type: record

**Table 7-90** local\_double\_write\_stat parameters

| Parameter | Type | Description   |
|-----------|------|---------------|
| node_name | text | Instance name |

| Parameter             | Type | Description                                                                                                                     |
|-----------------------|------|---------------------------------------------------------------------------------------------------------------------------------|
| curr_dwn              | int8 | Sequence number of the doublewrite file                                                                                         |
| curr_start_page       | int8 | Start page for restoring the doublewrite file                                                                                   |
| file_trunc_num        | int8 | Number of times that the doublewrite file is reused                                                                             |
| file_reset_num        | int8 | Number of reset times after the doublewrite file is full                                                                        |
| total_writes          | int8 | Total number of I/Os of the doublewrite file                                                                                    |
| low_threshold_writes  | int8 | Number of I/Os for writing doublewrite files with low efficiency (the number of I/O flushing pages at a time is less than 16)   |
| high_threshold_writes | int8 | Number of I/Os for writing doublewrite files with high efficiency (the number of I/O flushing pages at a time is more than 421) |
| total_pages           | int8 | Total number of pages that are flushed to the doublewrite file area                                                             |
| low_threshold_pages   | int8 | Number of pages that are flushed with low efficiency                                                                            |
| high_threshold_pages  | int8 | Number of pages that are flushed with high efficiency                                                                           |
| file_id               | int8 | ID of the current doublewrite file.                                                                                             |

- `local_single_flush_dw_stat()`  
Description: Displays the elimination of dual-write files on a single page in the instance.  
Return type: record
- `local_pagewriter_stat()`  
Description: Displays the page flushing information and checkpoint information of the current instance.  
Return type: record
- `local_redo_stat()`  
Description: Displays the replay status of the current standby instance.  
Return type: record  
Note: The returned replay status includes the current replay position and the replay position of the minimum restoration point.
- `local_recovery_status()`  
Description: Displays log flow control information about the primary and standby nodes.  
Return type: record
- `local_rto_status()`

Description: Displays log flow control information about the primary and standby nodes.

Return type: record

- `gs_wlm_switch_cgroup(sess_id int8, cgroup name)`

Description: Switches the Cgroup of a specified session.

Return type: record

- `comm_client_info()`

Description: Queries active client connections of a single node. For details about the returned result, see [COMM\\_CLIENT\\_INFO](#).

Return type: SETOF record

- `pg_get_flush_lsn()`

Description: Returns the position of the Xlog flushed from the current node.

Return type: text

- `pg_get_sync_flush_lsn()`

Description: Returns the position of the Xlog flushed by the majority on the current node.

Return type: text

- `pgxc_wlm_rebuild_user_resource_pool()`

Description: Rebuilds user and resource pool cache information. Only system administrators can execute this function.

Return type: Boolean

- `locktag_decode(locktag text)`

Description: Parses lock details from **locktag**.

Example:

```
gaussdb=# select locktag_decode('271b:0:0:0:6');
locktag_decode

locktype:transactionid, transactionid:10011
(1 row)
```

Return type: text

- `disable_conn(disconn_mode text, host text, port integer)`

Description: Specifies that the CM Agent processes commands delivered by the CM Server. When a DN is selected as the primary DN, it is configured to reject connections to all DNs, forcibly connect to a DN, or connect to all DNs in polling mode. Only the initial user and system administrator can call this function.

Return type: void

**Table 7-91** disable\_conn parameter description

| Parameter    | Type    | Description                                                                                                                                                                                                                                                                          |
|--------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| disconn_mode | text    | DN connection mode: <ul style="list-style-type: none"> <li>• <b>'prohibit_connection'</b>: rejects to connect to all DNs.</li> <li>• <b>'specify_connection'</b>: forcibly connects to a DN.</li> <li>• <b>'polling_connection'</b>: connects to all DNs in polling mode.</li> </ul> |
| host         | text    | IP address of the DN                                                                                                                                                                                                                                                                 |
| port         | integer | Port number of the DN                                                                                                                                                                                                                                                                |

- `db_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)`

Description: Obtains full SQL information about a cluster. The result can be queried only in the system database but cannot be queried in the user database.

Return type: record

**Table 7-92** db\_perf.get\_global\_full\_sql\_by\_timestamp parameter description

| Parameter       | Type                     | Description                                                                                                                                                        |
|-----------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| start_timestamp | timestamp with time zone | Start point of the SQL start time range.                                                                                                                           |
| end_timestamp   | timestamp with time zone | End point of the SQL start time range. An error is reported during function execution if <b>start_timestamp</b> is greater than or equal to <b>end_timestamp</b> . |

- `db_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)`

Description: Obtains cluster-level slow SQL information. The result can be queried only in the system database but cannot be queried in the user database.

Return type: record

**Table 7-93** db\_perf.get\_global\_slow\_sql\_by\_timestamp parameter description

| Parameter       | Type                     | Description                              |
|-----------------|--------------------------|------------------------------------------|
| start_timestamp | timestamp with time zone | Start point of the SQL start time range. |

| Parameter     | Type                     | Description                                                                                                                                                        |
|---------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| end_timestamp | timestamp with time zone | End point of the SQL start time range. An error is reported during function execution if <b>start_timestamp</b> is greater than or equal to <b>end_timestamp</b> . |

- statement\_detail\_decode(detail text, format text, pretty boolean)

Description: Parses the **details** column in a full or slow SQL statement. The result can be queried only in the system database but cannot be queried in the user database.

Return type: text

**Table 7-94** statement\_detail\_decode parameter description

| Parameter | Type    | Description                                                                                                                                                                                                                                                                                                                                 |
|-----------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| detail    | text    | Set of events generated by the SQL statement (unreadable).                                                                                                                                                                                                                                                                                  |
| format    | text    | Parsing output format. The value is <b>plaintext</b> .                                                                                                                                                                                                                                                                                      |
| pretty    | Boolean | Specifies whether to display the text in pretty format when <b>format</b> is set to <b>plaintext</b> . The options are as follows: <ul style="list-style-type: none"> <li>The value <b>true</b> indicates that \n is used to separate events.</li> <li>The value <b>false</b> indicates that events are separated by commas (,).</li> </ul> |

- pgxc\_get\_csn(tid)

Description: Returns the transaction submission sequence number (CSN) corresponding to a given transaction ID.

Return type: int8
- get\_global\_user\_transaction()

Description: Returns transaction information about each user on all nodes.

Return type: node\_name name, username name, commit\_counter bigint, rollback\_counter bigint, resp\_min bigint, resp\_max bigint, resp\_avg bigint, resp\_total bigint, bg\_commit\_counter bigint, bg\_rollback\_counter bigint, bg\_resp\_min bigint, bg\_resp\_max bigint, bg\_resp\_avg bigint, and bg\_resp\_total bigint
- pg\_collation\_for()

Description: Returns the sorting rule corresponding to the input parameter string.

Parameter: any (Explicit type conversion is required for constants.)

Return type: text

- `pgxc_unlock_for_sp_database(name Name)`  
 Description: Releases a specified database lock.  
 Parameter: database name  
 Return type: Boolean
- `pgxc_lock_for_sp_database(name Name)`  
 Description: Locks a specified database.  
 Parameter: database name  
 Return type: Boolean
- `pgxc_unlock_for_transfer(name Name)`  
 Description: Releases the lock used for data transmission (data redistribution).  
 Parameter: database name  
 Return type: Boolean
- `pgxc_lock_for_transfer(name Name)`  
 Description: Locks the database for data transmission (data redistribution).  
 Parameter: database name  
 Return type: Boolean
- `gs_comm_proxy_thread_status()`  
 Description: Collects statistics on data packets sent and received by the proxy communications library **comm\_proxy** when a user-mode network is configured for the cluster. This function is not supported in the current version.  
 Parameter: nan  
 Return type: record
- `gs_catalog_attribute_records()`  
 Description: Returns the definition of each field in a specified system catalog. Only common system catalogs whose OIDs are less than 10000 are supported. Indexes and TOAST tables are not supported.  
 Parameter: OID of the system catalog  
 Return type: record
- `dynamic_func_control(scope text, function_name text, action text, "{params}" text[])`  
 Description: Dynamically enables built-in functions. Currently, only full SQL statements can be dynamically enabled.  
 Return type: record

**Table 7-95** Parameter description of `dynamic_func_control`

| Parameter     | Type | Description                                                                                                          |
|---------------|------|----------------------------------------------------------------------------------------------------------------------|
| scope         | text | Scope where the function is to be dynamically enabled. Currently, only <b>GLOBAL</b> and <b>LOCAL</b> are supported. |
| function_name | text | Function name. Currently, only <b>STMT</b> is supported.                                                             |

| Parameter | Type   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| action    | text   | <p>When <b>function_name</b> is set to <b>STMT</b>, the value of <b>action</b> can only be <b>TRACK</b>, <b>UNTRACK</b>, <b>LIST</b>, or <b>CLEAN</b>.</p> <ul style="list-style-type: none"> <li>• <b>TRACK</b>: records the full SQL information of normalized SQL statements.</li> <li>• <b>UNTRACK</b>: cancels the recording of full SQL information of normalized SQL statements.</li> <li>• <b>LIST</b>: lists normalized SQL information that is recorded in the current track.</li> <li>• <b>CLEAN</b>: cleans normalized SQL information that is recorded in the current track.</li> </ul> |
| params    | text[] | <p>When <b>function_name</b> is set to <b>STMT</b>, the parameters corresponding to different actions are set as follows:</p> <ul style="list-style-type: none"> <li>• <b>TRACK</b>: '{"Normalized SQLID", "L0/L1/L2"}'</li> <li>• <b>UNTRACK</b>: '{"Normalized SQLID"}'</li> <li>• <b>LIST</b>: '{}'</li> <li>• <b>CLEAN</b>: '{}'</li> </ul>                                                                                                                                                                                                                                                      |

- `gs_parse_page_bypath(path text, blocknum bigint, relation_type text, read_memory boolean)`

Description: Parses a specified table page and returns the path for storing the parsed content.

Return type: text

Note: Only the SYSADMIN or OPRADMIN can execute this function.

**Table 7-96** gs\_parse\_page\_bypath parameters

| Parameter | Type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path      | text | <ul style="list-style-type: none"> <li>For an ordinary table, the relative path is <i>Tablespace name/Database OID/Relfilenode of the table (physical file name)</i>, for example, <b>base/16603/16394</b>.</li> <li>For the visibility map of an ordinary table, the relative path is <i>Tablespace name/Database OID/Visibility map of the ordinary table</i>. For example, <b>base/16603/16394_vm</b>.</li> <li>For clog files, the parsed content is stored in the <b>pg_clog</b> directory. For example: 000000000000.</li> <li>For csnlog files, the parsed content is stored in the <b>pg_csnlog</b> directory. For example: 000000000000.</li> <li>For undo record files, the relative path is <b>undo/UNDOPERSISTENCE/zondid.segno</b> in the <b>undo</b> directory. Example: undo/permanent/00000.0000009.</li> <li>For undo meta files, the relative path is <b>undo/UNDOPERSISTENCE/zondid.meta.segno</b> in the <b>undo</b> directory. Example: undo/permanent/00000.meta.0000004.</li> <li>You can run the <b>pg_relation_filepath(table_name text)</b> command to query the relative path of the table file. To obtain the path of the partitioned table, view the <b>pg_partition</b> system catalog and call <b>pg_partition_filepath(partition_oid)</b>.</li> <li>Valid path formats are as follows: <ul style="list-style-type: none"> <li>global/relNode</li> <li>base/dbNode/relNode</li> <li>pg_tblspc/spcNode/version_dir/dbNode/relNode</li> </ul> </li> </ul> |

| Parameter     | Type    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blocknum      | bigint  | <ul style="list-style-type: none"> <li>• <b>-1</b>: Information about all blocks (forcibly parsed from disks)</li> <li>• <b>0-MaxBlockNumber</b>: Information about the corresponding block</li> <li>• For B-tree/UB-tree indexes, <b>0</b> indicates the index meta-page.</li> <li>• For undo record files, the logical block number is used. The block number of the corresponding file is greater than or equal to <b>segno</b> x 128 and less than <b>(segno + 1) x 128</b>.</li> <li>• For undo meta files, the logical block number is used. The block number of the corresponding file is greater than or equal to <b>segno</b> x 4 and less than <b>(segno + 1) x 4</b>.</li> </ul>         |
| relation_type | text    | <ul style="list-style-type: none"> <li>• <b>heap</b>: Astore table</li> <li>• <b>uheap</b>: Ustore table</li> <li>• <b>btree</b>: B-tree index</li> <li>• <b>ubtree</b>: UB-tree index</li> <li>• <b>fsm</b>: free space of an Astore/Ustore heap table</li> <li>• <b>segment</b>: segment-page. This parameter is reserved and is not supported currently.</li> <li>• <b>vm</b>: visibility map of the Astore ordinary table</li> <li>• <b>clog</b> (commit log): transaction status log</li> <li>• <b>csnlog</b> (commit sequence number log): snapshot timestamp log</li> <li>• <b>undo_slot</b>: transaction slot information</li> <li>• <b>undo_record</b>: undo record information</li> </ul> |
| read_memory   | Boolean | <ul style="list-style-type: none"> <li>• <b>false</b>: The system parses the page from the disk file.</li> <li>• <b>true</b>: The system attempts to parse the page from the shared buffer. If the page does not exist in the shared buffer, the system parses the page from the disk file.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                              |

Example:

```
Parse the information of all pages in the B-tree index file.
Before calling the function, ensure that the file path exists based on the parameter description.
gaussdb=# select gs_parse_page_bypath('base/16603/16394', -1, 'btree', false);
gs_parse_page_bypath

/data_dir/1663_16603_16394_-1.page
```

```
(1 row)

Parse the visibility result of all blocks in the visibility map file.
gaussdb=# select gs_parse_page_bypath('base/12828/16771_vm', -1, 'vm', false);
 gs_parse_page_bypath

/data_dir/1663_12828_16771_-1_vm.page
(1 row)

Parse the commit log of block 0 in the Clog file.
gaussdb=# select gs_parse_page_bypath('000000000000', 0, 'clog', false);
 gs_parse_page_bypath

/data_dir/000000000000.clog
(1 row)
```

The following is an example of an exception error:

```
An error is reported when the value of the block number exceeds the value range.
gaussdb=# select gs_parse_page_bypath('base/12828/16777', -10, 'heap', false);
ERROR: Blocknum should be between -1 and 4294967294.
CONTEXT: referenced column: gs_parse_page_bypath
```

- `gs_xlogdump_lsn(start_lsn text, end_lsn text)`  
Description: Parses Xlogs within the specified LSN range and returns the path for storing the parsed content. You can use `pg_current_xlog_location()` to obtain the current Xlog position.  
Parameters: LSN start position and LSN end position  
Return type: text  
Note: Only the SYSADMIN or OPRADMIN can execute this function.
- `gs_xlogdump_xid(c_xid xid)`  
Description: Parses Xlogs of a specified XID and returns the path for storing the parsed content. You can use **`txid_current()`** to obtain the current XID.  
Parameter: XID  
Return type: text  
Note: Only the SYSADMIN or OPRADMIN can execute this function.
- `gs_xlogdump_tablepath(path text, blocknum bigint, relation_type text)`  
Description: Parses logs corresponding to a specified table page and returns the path for storing the parsed content.  
Return type: text  
Note: Only the SYSADMIN or OPRADMIN can execute this function.

**Table 7-97** gs\_xlogdump\_tablepath parameters

| Parameter     | Type   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path          | text   | <ul style="list-style-type: none"> <li>For an ordinary table, the relative path is <i>Tablespace name/Database OID/Relfilenode of the table (physical file name)</i>, for example, <b>base/16603/16394</b>.</li> <li>You can run the <b>pg_relation_filepath(table_name text)</b> command to query the relative path of the table file. To obtain the path of the partitioned table, view the <b>pg_partition</b> system catalog and call <b>pg_partition_filepath(partition_oid)</b>.</li> <li>Valid path formats are as follows: <ul style="list-style-type: none"> <li>global/relNode</li> <li>base/dbNode/relNode</li> <li>pg_tblspc/spcNode/version_dir/dbNode/relNode</li> </ul> </li> </ul> |
| blocknum      | bigint | <ul style="list-style-type: none"> <li><b>-1</b>: Information about all blocks (forcibly parsed from disks)</li> <li><b>0–MaxBlockNumber</b>: Information about the corresponding block</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| relation_type | text   | <ul style="list-style-type: none"> <li><b>heap</b>: Astore table</li> <li><b>btree</b>: B-tree index</li> <li><b>segment</b>: segment-page. This parameter is reserved and is not supported currently.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

- gs\_xlogdump\_parsepage\_tablepath(path text, blocknum bigint, relation\_type text, read\_memory boolean)

Description: Parses the specified table page and logs corresponding to the table page and returns the path for storing the parsed content. It can be regarded as one execution of **gs\_parse\_page\_bypath** and **gs\_xlogdump\_tablepath**. The prerequisite for executing this function is that the table file exists. To view logs of deleted tables, call **gs\_xlogdump\_tablepath**.

Return type: text

Note: Only the SYSADMIN or OPRADMIN can execute this function.

**Table 7-98** gs\_xlogdump\_parsepage\_tablepath parameters

| Parameter     | Type    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path          | text    | <ul style="list-style-type: none"> <li>For an ordinary table, the relative path is <i>Tablespace name/Database OID/Relfilenode of the table (physical file name)</i>, for example, <b>base/16603/16394</b>.</li> <li>You can run the <b>pg_relation_filepath(table_name text)</b> command to query the relative path of the table file. To obtain the path of the partitioned table, view the <b>pg_partition</b> system catalog and call <b>pg_partition_filepath(partition_oid)</b>.</li> <li>Valid path formats are as follows: <ul style="list-style-type: none"> <li>global/relNode</li> <li>base/dbNode/relNode</li> <li>pg_tblspc/spcNode/version_dir/dbNode/relNode</li> </ul> </li> </ul> |
| blocknum      | bigint  | <ul style="list-style-type: none"> <li><b>-1</b>: Information about all blocks (forcibly parsed from disks)</li> <li><b>0-MaxBlockNumber</b>: Information about the corresponding block</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| relation_type | text    | <ul style="list-style-type: none"> <li><b>heap</b>: Astore table</li> <li><b>btree</b>: B-tree index</li> <li><b>segment</b>: segment-page. This parameter is reserved and is not supported currently.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| read_memory   | Boolean | <ul style="list-style-type: none"> <li><b>false</b>: The system parses the page from the disk file.</li> <li><b>true</b>: The system attempts to parse the page from the shared buffer. If the page does not exist in the shared buffer, the system parses the page from the disk file.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                 |

- gs\_index\_recycle\_queue(Oid oid, int type, uint32 blkno)  
Description: Parses the UB-tree index recycling queue information.  
Return type: record

**Table 7-99** gs\_index\_recycle\_queue parameters

| Parameter | Type | Description                                                                                                                                                                   |
|-----------|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| oid       | Oid  | <ul style="list-style-type: none"> <li>Index file relfilenode, which can be queried using <b>select relfilenode from pg_class where relname='Index file name'</b>.</li> </ul> |

| Parameter | Type   | Description                                                                                                                                                                                                                                                  |
|-----------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| type      | int    | <ul style="list-style-type: none"> <li>• <b>0</b> indicates that the entire queue to be recycled is parsed.</li> <li>• <b>1</b> indicates that the entire empty page queue is parsed.</li> <li>• <b>2</b> indicates that a single page is parsed.</li> </ul> |
| blkno     | uint32 | ID of the recycling queue page. This parameter is valid only when <b>type</b> is set to <b>2</b> . The value of <b>blkno</b> ranges from 1 to 4294967294.                                                                                                    |

 **NOTE**

This function is not supported in the distributed version. An error message will be displayed if it is used in the distributed version.

- gs\_stat\_wal\_entrytable(int64 idx)

Description: Exports the content of the write-ahead log insertion status table in the Xlog.

Return type: record

**Table 7-100** gs\_stat\_wal\_entrytable parameters

| Category         | Parameter | Type   | Description                                                                                                                                                              |
|------------------|-----------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | idx       | int64  | <ul style="list-style-type: none"> <li>• <b>-1</b>: queries all elements in an array.</li> <li>• <b>0-Maximum value</b>: content of a specific array element.</li> </ul> |
| Output parameter | idx       | uint64 | Records the indexes in the corresponding array.                                                                                                                          |
| Output parameter | endsln    | uint64 | Records the LSN label.                                                                                                                                                   |
| Output parameter | lrc       | int32  | Records the corresponding LRC.                                                                                                                                           |

| Category         | Parameter | Type   | Description                                                                                                                                                                                                      |
|------------------|-----------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Output parameter | status    | uint32 | Specifies whether the Xlog corresponding to the current entry has been completely copied to the WAL buffer. <ul style="list-style-type: none"> <li>• <b>0</b>: not copied</li> <li>• <b>1</b>: copied</li> </ul> |

- `gs_walwriter_flush_position()`  
Description: Outputs the refresh position of write-ahead logs.  
Return type: record

**Table 7-101** `gs_walwriter_flush_position` parameters

| Category         | Parameter               | Type   | Description                                                                                                                                                                |
|------------------|-------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Output parameter | last_flush_status_entry | int32  | Index obtained after the Xlog flushes the tblEntry of the last flushed disk.                                                                                               |
| Output parameter | last_scanned_lrc        | int32  | LRC obtained after the Xlog flushes the last tblEntry scanned last time.                                                                                                   |
| Output parameter | curr_lrc                | int32  | Latest LRC usage in the WALInsertStatusEntry status table. The LRC indicates the LRC value corresponding to the WALInsertStatusEntry when the next Xlog record is written. |
| Output parameter | curr_byte_pos           | uint64 | The latest Xlog position after the Xlog is written to the WAL file, which is also the next Xlog insertion point.                                                           |
| Output parameter | prev_byte_size          | uint32 | Length of the previous Xlog record.                                                                                                                                        |
| Output parameter | flush_result            | uint64 | Position of the current global Xlog flush.                                                                                                                                 |

| Category         | Parameter            | Type   | Description                                                                                  |
|------------------|----------------------|--------|----------------------------------------------------------------------------------------------|
| Output parameter | send_result          | uint64 | Xlog sending position on the current host.                                                   |
| Output parameter | shm_rqst_write_pos   | uint64 | The write position of the LogwrtRqst request in the XLogCtl recorded in the shared memory.   |
| Output parameter | shm_rqst_flush_pos   | uint64 | The flush position of the LogwrtRqst request in the XLogCtl recorded in the shared memory.   |
| Output parameter | shm_result_write_pos | uint64 | The write position of the LogwrtResult request in the XLogCtl recorded in the shared memory. |
| Output parameter | shm_result_flush_pos | uint64 | The flush position of the LogwrtResult request in the XLogCtl recorded in the shared memory. |
| Output parameter | curr_time            | text   | Current time.                                                                                |

- `gs_walwriter_flush_stat(int operation)`  
Description: Collects statistics on the frequency of writing and synchronizing write-ahead logs, data volume, and Xlog file information.  
Return type: record

**Table 7-102** `gs_walwriter_flush_stat` parameters

| Category        | Parameter | Type | Description                                                                                                                                                                                                                                         |
|-----------------|-----------|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter | operation | int  | <ul style="list-style-type: none"> <li>• <b>-1</b>: Disables the statistics function. (Default value)</li> <li>• <b>0</b>: Enable the statistics function.</li> <li>• <b>1</b>: Query statistics.</li> <li>• <b>2</b>: Reset statistics.</li> </ul> |

| Category         | Parameter                    | Type   | Description                                                        |
|------------------|------------------------------|--------|--------------------------------------------------------------------|
| Output parameter | write_times                  | uint64 | Number of times that the Xlog calls the write API.                 |
| Output parameter | sync_times                   | uint64 | Number of times that the Xlog calls the sync API.                  |
| Output parameter | total_xlog_sync_bytes        | uint64 | Total number of backend thread requests for writing data to Xlogs. |
| Output parameter | total_actual_xlog_sync_bytes | uint64 | Total number of Xlogs that call the sync API for disk flushing.    |
| Output parameter | avg_write_bytes              | uint32 | Number of Xlogs written each time the XLogWrite API is called.     |
| Output parameter | avg_actual_write_bytes       | uint32 | Number of Xlogs written each time the write API is called.         |
| Output parameter | avg_sync_bytes               | uint32 | Average number of Xlogs for each synchronization request.          |
| Output parameter | avg_actual_sync_bytes        | uint32 | Actual Xlog amount of disk flushing by calling sync each time.     |
| Output parameter | total_write_time             | uint64 | Total time of calling the write operation (unit: $\mu$ s).         |
| Output parameter | total_sync_time              | uint64 | Total time for calling the sync API (unit: $\mu$ s).               |

| Category         | Parameter            | Type   | Description                                                       |
|------------------|----------------------|--------|-------------------------------------------------------------------|
| Output parameter | avg_write_time       | uint32 | Average time for calling the write API each time (unit: $\mu$ s). |
| Output parameter | avg_sync_time        | uint32 | Average time for calling the sync API each time (unit: $\mu$ s).  |
| Output parameter | curr_init_xlog_segno | uint64 | ID of the latest Xlog segment file.                               |
| Output parameter | curr_open_xlog_segno | uint64 | ID of the Xlog segment file that is being written.                |
| Output parameter | last_reset_time      | text   | Time when statistics were last collected.                         |
| Output parameter | curr_time            | text   | Current time.                                                     |

- pg\_ls\_tmpdir()

Description: Returns the name, size, and last modification time of each file in the temporary directory (**pgsql\_tmp**) of the default tablespace.

Parameter: nan

Return type: record

Note: Only the SYSADMIN or MONADMIN can execute this function.

| Category         | Parameter    | Type       | Description                  |
|------------------|--------------|------------|------------------------------|
| Output parameter | name         | text       | File name.                   |
| Output parameter | size         | int8       | File size (unit: byte).      |
| Output parameter | modification | timestampz | Last file modification time. |

- `pg_ls_tmpdir(oid)`

Description: Returns the name, size, and last modification time of each file in the temporary directory (**pgsql\_tmp**) of the specified tablespace.

Parameter: oid

Return type: record

Note: Only the SYSADMIN or MONADMIN can execute this function.

| Category         | Parameter    | Type        | Description                  |
|------------------|--------------|-------------|------------------------------|
| Input parameter  | oid          | oid         | Tablespace ID.               |
| Output parameter | name         | text        | File name.                   |
| Output parameter | size         | int8        | File size (unit: byte).      |
| Output parameter | modification | timestamptz | Last file modification time. |

- `pg_ls_waldir()`

Description: Returns the name, size, and last modification time of each file in the WAL directory.

Parameter: nan

Return type: record

Note: Only the SYSADMIN or MONADMIN can execute this function.

| Category         | Parameter    | Type        | Description                  |
|------------------|--------------|-------------|------------------------------|
| Output parameter | name         | text        | File name.                   |
| Output parameter | size         | int8        | File size (unit: byte).      |
| Output parameter | modification | timestamptz | Last file modification time. |

- `gs_write_term_log(void)`

Description: Writes a log to record the current **term** value of a DN. The standby DN returns **false**. After the data is successfully written to the primary DN, **true** is returned.

Return type: Boolean

- `gs_stat_space(bool init)`

Description: Queries the status of extended pages when the INSERT operation is performed on the Ustore table.

Return type: record

| Category         | Parameter          | Type | Description                                                                                                    |
|------------------|--------------------|------|----------------------------------------------------------------------------------------------------------------|
| Input parameter  | init               | bool | Specifies whether to reset the statistics.                                                                     |
| Output parameter | access_func        | int8 | Total number of access times of the relation_get_buffer_for_utuple API.                                        |
| Output parameter | cache_blk          | int8 | Number of times that the relation_get_buffer_for_utuple API obtains buffers.                                   |
| Output parameter | cache_succ         | int8 | Number of times that the relation_get_buffer_for_utuple API successfully obtains buffers.                      |
| Output parameter | nblk_first         | int8 | Number of times that relation_get_buffer_for_utuple obtains <b>nblocks-1</b> for the first time.               |
| Output parameter | nblk_first_succ    | int8 | Number of times that relation_get_buffer_for_utuple obtains <b>nblocks-1</b> successfully for the first time.  |
| Output parameter | nblk_second        | int8 | Number of times that relation_get_buffer_for_utuple obtains <b>nblocks-1</b> for the second time.              |
| Output parameter | nblk_second_succ   | int8 | Number of times that relation_get_buffer_for_utuple obtains <b>nblocks-1</b> successfully for the second time. |
| Output parameter | fsm_first          | int8 | Number of times that FSM is accessed for the first time.                                                       |
| Output parameter | fsm_first_success  | int8 | Number of times that FSM is accessed successfully for the first time.                                          |
| Output parameter | fsm_rewrite        | int8 | Number of FSM writeback times.                                                                                 |
| Output parameter | fsm_second         | int8 | Number of times that FSM is accessed for the second time.                                                      |
| Output parameter | fsm_second_success | int8 | Number of times that FSM is accessed successfully for the second time.                                         |



page cleaning mechanism may be faulty. If the value of **con\_extend\_time** is too large, the Ustore concurrent page extension takes a long time.

- `gs_redo_upage(directory_path text, backup_path text, blocknum bigint, relation_type text, xlog_path text, lsn text)`

Description:Redoes a specific Ustore data page that is backed up to a specified LSN and verifies the page in this process. If a damaged page is detected, the page is flushed to the disk and the disk flushing path, page LSN, and damage information is returned; otherwise, the page is redone to the specified LSN and flushed to the disk, and the result is returned. Only system administrators or O&M administrators can execute this function.

Return type: record

| Category         | Parameter       | Type   | Description                                                                                                                                                                                                                                                            |
|------------------|-----------------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | directory_path  | text   | Specifies the directory for storing the backup file.                                                                                                                                                                                                                   |
| Input parameter  | backup_path     | text   | Relative path of the backup table file, which is combined with the directory where the backup file is located to form the complete path of the table file, for example, <b>base/15635/12488</b> . If the backup file does not exist, set this parameter to null.       |
| Input parameter  | blocknum        | bigint | <b>0</b> to <i>MaxBlockNumber</i> : block number of the corresponding page.                                                                                                                                                                                            |
| Input parameter  | relation_type   | text   | <ul style="list-style-type: none"> <li>• <b>uheap</b>: Ustore data page</li> <li>• <b>ubtree</b>: Ustore index page</li> <li>• <b>indexurq</b>: Urq page</li> <li>• <b>undo_record</b>: Undo record page</li> <li>• <b>undo_slot</b>: Transaction slot page</li> </ul> |
| Input parameter  | xlog_path       | text   | Absolute path of the archive log directory.                                                                                                                                                                                                                            |
| Input parameter  | lsn             | text   | The LSN consists of two hexadecimal numbers (32 bits each), which are separated by a slash (/), for example, <b>2/962D1DF8</b> . If the value is <b>0</b> , the latest version is used.                                                                                |
| Output parameter | output_filename | text   | Path and name of the file to be flushed to the disk.                                                                                                                                                                                                                   |
| Output parameter | output_lsn      | text   | LSN of the last page redo.                                                                                                                                                                                                                                             |
| Output parameter | corruption_desc | text   | Page damage description.                                                                                                                                                                                                                                               |

- gs\_xlogdump\_bylastlsn(last\_lsn text, blocknum bigint, relation\_type text)**  
 Description: Inputs a page LSN and block number, parses the WAL corresponding to the LSN, obtains the last LSN of the corresponding block number, continues parsing until the last LSN is 0 or the WAL of an earlier version has been reused and recycled, and flushes the parsed log to a specified path. Only system administrators or O&M administrators can execute this function. This system function cannot be called by the standby node.

Return type: text

| Category         | Parameter       | Type   | Description                                                                                                                                                                                        |
|------------------|-----------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | last_lsn        | text   | Parses the LSN of a specified page in hexadecimal notation, for example, 12BA/32CDEDDD. The LSN can be obtained using a page parsing tool (such as <b>gs_parse_page_bypath</b> ).                  |
| Input parameter  | blocknum        | bigint | Specifies the logical block number of a page.<br>Value range: <b>-1</b> to <i>MaxBlockNumber</i> .<br>If the block number is set to <b>-1</b> , the default block number is obtained from the WAL. |
| Input parameter  | relation_type   | text   | Specifies the type of the page to be parsed.<br>Valid value: <b>uheap</b> , <b>ubtree</b> , <b>heap</b> , <b>btree</b> , <b>undo_record</b> , and <b>undo_slot</b> .                               |
| Output parameter | output_filepath | text   | Specifies the path for flushing WAL parsing results to disks.                                                                                                                                      |

**Example:**

```
Obtain the page LSN.
Before calling the function, ensure that the file path exists based on the parameter description.
gaussdb=# select * from gs_parse_page_bypath('base/15833/16768', 0, 'uheap', false);
output_filepath

/data1/database/cluster/primary/data/1663_15833_16768_0.page
(1 row)
gaussdb=# select * from gs_xlogdump_bylastlsn('0/4593570', -1, 'uheap');
output_filepath

/data1/database/cluster/primary/data/pg_log/dump/4593570_-1.xlog
(1 row)
gaussdb=# select * from gs_xlogdump_bylastlsn('0/4593570', 0, 'ubtree');
ERROR: The input lsn 0/4593570 related xlog is not ubtree.
```

- dbe\_perf.get\_full\_sql\_by\_parent\_id\_and\_timestamp(parent\_id bigint, start\_timestamp timestamp with time zone, end\_timestamp timestamp with time zone)**

Description: Obtains the records of database-level full SQL statements and their substatements that execute a stored procedure in a specified period

based on **parent\_id**. The result can be queried only in the system database but cannot be queried in the user database.

Return type: record

| Parameter       | Type                     | Description                                                              |
|-----------------|--------------------------|--------------------------------------------------------------------------|
| parent_id       | bigint                   | Specifies the unique_sql_id of the statement calling a stored procedure. |
| start_timestamp | timestamp with time zone | Start point of the SQL start time range.                                 |
| end_timestamp   | timestamp with time zone | End point of the SQL start time range.                                   |

**Example:**

```
gaussdb=# CREATE TABLE test(a int,b int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO test values(1,1);
INSERT 0 1
gaussdb=# CREATE PROCEDURE mypro1() as num int;
gaussdb$# begin
gaussdb$# INSERT INTO test values(2,2);
gaussdb$# DELETE FROM test where a = 2;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
```

Enable the parameter to trace the substatements of the stored procedure.

```
gaussdb=# SET instr_unique_sql_track_type = 'all';
SET
```

Enable the parameter. Full statement records are generated in the db\_perf.statement\_history catalog.

```
gaussdb=# SET track_stmt_stat_level = 'L0,L0';
SET
```

```
gaussdb=# CALL mypro1();
mypro1
```

(1 row)

```
gaussdb=# SET track_stmt_stat_level = 'off,L0';
SET
```

```
gaussdb=# SET instr_unique_sql_track_type = 'top';
SET
```

Query key information, which is used as a function parameter.

```
gaussdb=# SELECT query,unique_query_id,start_time,finish_time FROM db_perf.statement_history;
 query | unique_query_id | start_time | finish_time
-----+-----+-----+-----
set track_stmt_stat_level = 'L0,L0'; | 636388010 | 2023-06-02 17:40:49.176155+08 | 2023-06-02 17:40:49.176543+08
call mypro1(); | 536458473 | 2023-06-02 17:40:59.028144+08 | 2023-06-02 17:40:59.032027+08
delete from test where a = ? | 583323884 | 2023-06-02 17:40:59.029955+08 | 2023-06-02 17:40:59.031577+08
insert into test values(?,?) | 769279931 | 2023-06-02 17:40:59.029219+08 | 2023-06-02 17:40:59.029947+08
```

```
(4 rows)

Use unique_query_id, start time, and end time of the outer query statement as parameters to query
information about the specified stored procedure and its substatements within the period.
gaussdb=# SELECT query FROM
dbe_perf.get_full_sql_by_parent_id_and_timestamp(536458473,'2023-06-02
17:40:59.028144+08','2023-06-02 17:40:59.032027+08');
query

call mypro1();
delete from test where a = ?
insert into test values(?,?)
(3 rows)

Drop the table.
gaussdb=# DROP TABLE test;
DROP TABLE

Drop the stored procedure.
gaussdb=# DROP PROCEDURE mypro1;
DROP PROCEDURE
```

- `gs_index_dump_read(int8 reset, text out_type)`

Description: Queries the buffer read information generated in the cyclic queue when an index is used to obtain a new page. The buffer read information traverses leaf pages from left to right using the same key as the index page.

Return type: record

| Category         | Parameter   | Type | Description                                                                                                                                                                                                                                                |
|------------------|-------------|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input parameter  | reset       | int8 | <ul style="list-style-type: none"> <li>• <b>0</b>: The statistics are reset to the initial value 0 and the statistics are collected again.</li> <li>• <b>1</b>: The current statistics are displayed.</li> </ul>                                           |
| Input parameter  | out_type    | text | <ul style="list-style-type: none"> <li>• <b>urq</b>: outputs statistics about cyclic queues.</li> <li>• <b>ubtree</b>: outputs statistics on index pages.</li> <li>• <b>all</b>: All statistics about cyclic queues and index pages are output.</li> </ul> |
| Output parameter | relfilenode | oid  | Index relfilenode corresponding to the maximum buffer read value.                                                                                                                                                                                          |
| Output parameter | max_count   | int8 | Maximum buffer read value.                                                                                                                                                                                                                                 |
| Output parameter | ave_count   | int8 | Average buffer read value.                                                                                                                                                                                                                                 |

 **NOTE**

- Currently, this API supports only Ustore index tables.
- When this API is executed, the reset operation is performed to clear all records and set all records to 0. If you query the information again, the query result is always 0 until the information is collected next time. The following are query examples:

```
gaussdb=# SELECT * FROM gs_index_dump_read(0, 'all');
relfilenode | max_count | ave_count
-----+-----+-----
| | |
(1 row)
gaussdb=# SELECT * FROM gs_index_dump_read(1, 'all');
relfilenode | max_count | ave_count
-----+-----+-----
0 | | 0
0 | | 0
(2 rows)
```

### 7.5.25.13 SQL Statement Concurrency Control Function

- `gs_add_workload_rule(rule_type, rule_name, databases, start_time, end_time, max_workload, option_val)`

Description: Creates an SQL statement concurrency control rule. Users must have the SYSADMIN permission. This function can be executed only on the CN.

Parameters: For details, see [Table 7-103](#).

Return type: int8

**Table 7-103** gs\_add\_workload\_rule parameters

| Parameter | Type | Description                                                                                   | Value Range                                                                                                                                                                                                                                                              |
|-----------|------|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rule_type | text | Type of the concurrency control rule, which is case-insensitive.                              | "sqlid": Concurrency control is based on the unique SQL ID.<br>"select", "insert", "update", "delete", and "merge": Concurrency control is based on the query type and keyword.<br>"resource": Instance-level concurrency control is based on the system resource usage. |
| rule_name | name | Name of a concurrency control rule, which is used to search for the concurrency control rule. | Any character string or <b>NULL</b> .                                                                                                                                                                                                                                    |

| Parameter    | Type       | Description                                                                                               | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------|------------|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| databases    | name[]     | Array of database names for which the concurrency control rule takes effect. The value is case-sensitive. | List of names of created databases. The value can be <b>NULL</b> , indicating that the configuration takes effect in all databases.<br><br>Currently, the database list takes effect only when <b>rule_type</b> is set to a query type because a unique SQL ID is bound to a database and belongs to only one database. The concurrency control rules based on resource usage take effect for instances, that is, all databases. |
| start_time   | timestampz | Start time of a concurrency control rule.                                                                 | The value can be <b>NULL</b> , indicating that it takes effect from now on.                                                                                                                                                                                                                                                                                                                                                      |
| end_time     | timestampz | End time of a concurrency control rule.                                                                   | The value can be <b>NULL</b> , indicating that the rule is always effective.                                                                                                                                                                                                                                                                                                                                                     |
| max_workload | int8       | Maximum number of concurrent requests set in a concurrency control rule.                                  | -                                                                                                                                                                                                                                                                                                                                                                                                                                |

| Parameter  | Type   | Description                                                   | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------|--------|---------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| option_val | text[] | Supplementary information about the concurrency control rule. | <p>It matches <b>rule_type</b>. The matching relationship is as follows:</p> <ul style="list-style-type: none"> <li>• <b>"sqlid"</b>: specifies the unique ID of the SQL statement whose concurrency is to be controlled and slow SQL control rule. The format is '{id=1234, time_limit=100, max_execute_time=500, max_iops=1}', in which <b>id</b> indicates the unique SQL ID and is required. You can obtain it from the <code>dbe_perf.statement</code> or <code>pg_stat_activity</code> view. Others are optional. For details about their meanings, see section "Hint for Setting Slow SQL Control Rules."</li> <li>• <b>"select", "insert", "update", "delete", and "merge"</b>: specify the keyword sequence for concurrency control, which are case-insensitive and can be <b>NULL</b>.</li> <li>• <b>"resource"</b>: specifies the resource threshold for traffic limiting. The format is '{cpu-80, memory-70}'. No matter what the CPU and memory thresholds are set, the value <b>0</b> is used.</li> </ul> |

Example:

```

gaussdb=# select gs_add_workload_rule('sqlid', 'rule for one query', '{}', now(), NULL, 20,
'{id=32413214}');
gs_add_workload_rule

1
(1 row)
gaussdb=# create database db1;
gaussdb=# create database db2;
gaussdb=# select gs_add_workload_rule('select', 'rule for select', '{db1, db2}', NULL, NULL, 100, '{tb1,
tb2}');
gs_add_workload_rule

2
(1 row)
gaussdb=# select gs_add_workload_rule('resource', 'rule for resource', '{}', NULL, NULL, 20, '{cpu-80}');
gs_add_workload_rule

```

```

3
(1 row)
gaussdb=# drop database db1;
DROP DATABASE
gaussdb=# drop database db2;
DROP DATABASE
```

- `gs_update_workload_rule(rule_id, rule_name, databases, start_time, end_time, max_workload, option_val)`

Description: To update an SQL statement concurrency control rule, users need to reset all parameters instead of only some parameters. Users must have the SYSADMIN permission. This function can be executed only on the CN.

Parameters: For details, see [Table 7-104](#).

Return type: Boolean

**Table 7-104** `gs_update_workload_rule` parameters

| Parameter               | Type                     | Description                                                                                               | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------|--------------------------|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>rule_id</code>    | <code>int8</code>        | ID of the concurrency control rule to be updated.                                                         | -                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>rule_name</code>  | <code>name</code>        | Name of a concurrency control rule, which is used to search for the concurrency control rule.             | Any character string or <b>NULL</b> .                                                                                                                                                                                                                                                                                                                                                                                            |
| <code>databases</code>  | <code>name[]</code>      | Array of database names for which the concurrency control rule takes effect. The value is case-sensitive. | List of names of created databases. The value can be <b>NULL</b> , indicating that the configuration takes effect in all databases.<br><br>Currently, the database list takes effect only when <b>rule_type</b> is set to a query type because a unique SQL ID is bound to a database and belongs to only one database. The concurrency control rules based on resource usage take effect for instances, that is, all databases. |
| <code>start_time</code> | <code>timestamptz</code> | Start time of a concurrency control rule.                                                                 | The value can be <b>NULL</b> , indicating that it takes effect from now on.                                                                                                                                                                                                                                                                                                                                                      |

| Parameter    | Type       | Description                                                              | Value Range                                                                  |
|--------------|------------|--------------------------------------------------------------------------|------------------------------------------------------------------------------|
| end_time     | timestampz | End time of a concurrency control rule.                                  | The value can be <b>NULL</b> , indicating that the rule is always effective. |
| max_workload | int8       | Maximum number of concurrent requests set in a concurrency control rule. | -                                                                            |

| Parameter  | Type   | Description                                                   | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------|--------|---------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| option_val | text[] | Supplementary information about the concurrency control rule. | <p>It matches <b>rule_type</b>. The matching relationship is as follows:</p> <ul style="list-style-type: none"> <li>• <b>"sqlid"</b>: specifies the unique ID of the SQL statement whose concurrency is to be controlled and slow SQL control rule. The format is '{id=1234, time_limit=100, max_execute_time=500, max_iops=1}', in which <b>id</b> indicates the unique SQL ID and is required. You can obtain it from the <code>dbe_perf.statement</code> or <code>pg_stat_activity</code> view. Others are optional. For details about their meanings, see section "Hint for Setting Slow SQL Control Rules."</li> <li>• <b>"select", "insert", "update", "delete", and "merge"</b>: specify the keyword sequence for concurrency control, which are case-insensitive and can be <b>NULL</b>.</li> <li>• <b>"resource"</b>: specifies the resource threshold for triggering instance-level concurrency control. The value is in the format of '{cpu-80, memory-70}', indicating the OS resource threshold for triggering instance-level concurrency control. The value can be <b>NULL</b>, indicating that concurrency control is performed regardless of the resource usage.</li> </ul> |

Example:

```
gaussdb=# create database db1;
gaussdb=# select gs_update_workload_rule(2, 'rule for select 2', '{db1}', now(), NULL, 50, '{tb1}');
gs_update_workload_rule

t
(1 row)
```

```
gaussdb=# drop database db1;
DROP DATABASE
```

- **gs\_delete\_workload\_rule(rule\_id)**

Description: Deletes an SQL statement concurrency control rule. Users must have the SYSADMIN permission. This function can be executed only on the CN.

Parameter: **rule\_id** indicates the ID of the concurrency control rule to be updated. The type is int8.

Return type: Boolean

Example:

```
gaussdb=# select gs_delete_workload_rule(3);
gs_delete_workload_rule

t
(1 row)
```

- **gs\_get\_workload\_rule\_stat(rule\_id)**

Description: Queries the number of times that SQL statements are blocked by SQL statement concurrency control rules. Users must have the SYSADMIN permission. This function can be executed only on the CN.

Parameter: **rule\_id** indicates the ID of the concurrency control rule to be queried. The type is int8. You can set **rule\_id** to **-1**, indicating that all SQL statement concurrency control rules are queried.

Return type:

| Name           | Type | Description                                                                         |
|----------------|------|-------------------------------------------------------------------------------------|
| rule_id        | int8 | ID of the SQL statement concurrency control rule.                                   |
| validate_count | int8 | Number of SQL statements intercepted by the SQL statement concurrency control rule. |

Example:

```
gaussdb=# select * from gs_get_workload_rule_stat(1);
rule_id | validate_count
-----+-----
1 | 0
(1 row)
gaussdb=# select * from gs_get_workload_rule_stat(-1);
rule_id | validate_count
-----+-----
1 | 0
2 | 0
(2 rows)
```

## 7.5.26 Statistics Information Functions

Statistics information functions are divided into the following two categories: functions that access databases, using the OID of each table or index in a database to mark the database for which statistics are generated; functions that access servers, identified by the server process ID, whose value ranges from 1 to the number of currently active servers.

- `pg_stat_get_db_conflict_tablespace(oid)`  
Description: Specifies the number of queries canceled due to a conflict between the restored tablespace and the deleted tablespace in the database.  
Return type: `bigint`
- `pg_control_group_config()`  
Description: Prints Cgroup configurations on the current node. Only users with the SYSADMIN permission can execute this function.  
Return type: `record`
- `pg_stat_get_db_stat_reset_time(oid)`  
Description: Specifies the most recent time when database statistics were reset. It is initialized to the system time during the first connection to each database. The reset time is updated when you call `pg_stat_reset` on the database and execute `pg_stat_reset_single_table_counters` against any table or index in it.  
Return type: `timestampz`
- `pg_stat_get_function_total_time(oid)`  
Description: Specifies the total wall clock time spent in the function, in microseconds. The time spent on this function calling other functions is included.  
Return type: `bigint`
- `pg_stat_get_xact_tuples_returned(oid)`  
Description: Specifies the number of rows read through sequential scans when the parameter is a table in the current transaction or the number of index entries returned when the parameter is an index.  
Return type: `bigint`
- `pg_stat_get_xact_numscans(oid)`  
Description: Specifies the number of sequential scans performed when the parameter is a table in the current transaction or the number of index scans performed when the parameter is an index.  
Return type: `bigint`
- `pg_stat_get_xact_blocks_fetched(oid)`  
Description: Specifies the number of disk block fetch requests for a table or an index in the current transaction.  
Return type: `bigint`
- `pg_stat_get_xact_blocks_hit(oid)`  
Description: Specifies the number of disk block fetch requests for tables or indexes found in cache in the current transaction.  
Return type: `bigint`
- `pg_stat_get_xact_function_calls(oid)`  
Description: Specifies the number of times the function is called in the current transaction.  
Return type: `bigint`
- `pg_stat_get_xact_function_self_time(oid)`

Description: Specifies the time spent only on this function in the current transaction. The time spent on other function call inside this function is not included.

Return type: bigint

- `pg_stat_get_xact_function_total_time(oid)`

Description: Specifies the total wall clock time spent on this function in the current transaction, in microseconds. The time spent on other function call inside this function is included.

Return type:

- `pg_lock_status()`

Description: Queries information about locks held by open transactions. All users can execute this function.

Return type: For details, see the return result of [PG\\_LOCKS](#), which is obtained by querying this function.

- `gs_lwlock_status()`

Description: Queries information about all lightweight locks in the database system, including lock waiting and lock holding information. All users can execute this function.

Return type: setofrecord

- `pg_stat_get_wal_senders()`

Description: Queries walsender information on the primary node.

Return type: setofrecord

The following table describes return columns.

**Table 7-105** Return column description

| Column        | Type                     | Description                             |
|---------------|--------------------------|-----------------------------------------|
| pid           | bigint                   | Thread ID of the WAL sender             |
| sender_pid    | integer                  | Lightweight thread ID of the WAL sender |
| local_role    | text                     | Type of the primary node                |
| peer_role     | text                     | Type of the standby node                |
| peer_state    | text                     | Status of the standby node              |
| state         | text                     | Status of the WAL sender                |
| catchup_start | timestamp with time zone | Startup time of a catchup task          |
| catchup_end   | timestamp with time zone | End time of a catchup task              |

| Column                     | Type | Description                            |
|----------------------------|------|----------------------------------------|
| sender_sent_location       | text | Sending position of the primary node   |
| sender_write_location      | text | Writing position of the primary node   |
| sender_flush_location      | text | Flushing position of the primary node  |
| sender_replay_location     | text | Redo position of the primary node      |
| receiver_received_location | text | Receiving position of the standby node |
| receiver_write_location    | text | Writing position of the standby node   |
| receiver_flush_location    | text | Flushing position of the standby node  |
| receiver_replay_location   | text | Redo position of the standby node      |
| sync_percent               | text | Synchronization percentage             |
| sync_state                 | text | Synchronization status                 |
| sync_priority              | text | Priority of synchronous replication    |
| sync_most_available        | text | Maximum availability mode              |
| channel                    | text | Channel information of the WAL sender  |

- pgxc\_get\_senders\_catchup\_time()**  
 Description: Queries whether a standby DN in the log catchup state exists in the CN instance query cluster and details about the log catchup state.  
 Return type: setofrecord
- pg\_stat\_get\_stream\_replications()**  
 Description: Queries the primary/standby replication status.  
 Return type: setofrecord  
 The following table describes return values.

**Table 7-106** Return value description

| Return Parameter | Type | Description |
|------------------|------|-------------|
| local_role       | text | Local role  |

| Return Parameter   | Type    | Description           |
|--------------------|---------|-----------------------|
| static_connections | integer | Connection statistics |
| db_state           | text    | Database status       |
| detail_information | text    | Detailed information  |

- `pg_stat_get_db_numbackends(oid)`  
Description: Specifies the number of active server processes for a database.  
Return type: integer
- `pg_stat_get_db_xact_commit(oid)`  
Description: Specifies the number of transactions committed in a database.  
Return type: bigint
- `pg_stat_get_db_xact_rollback(oid)`  
Description: Specifies the number of transactions rolled back in a database.  
Return type: bigint
- `pg_stat_get_db_blocks_fetched(oid)`  
Description: Specifies the number of disk blocks fetch requests for a database.  
Return type: bigint
- `pg_stat_get_db_blocks_hit(oid)`  
Description: Specifies the number of disk block fetch requests found in cache for a database.  
Return type: bigint
- `pg_stat_get_db_tuples_returned(oid)`  
Description: Specifies the number of tuples returned for a database.  
Return type: bigint
- `pg_stat_get_db_tuples_fetched(oid)`  
Description: Specifies the number of tuples fetched for a database.  
Return type: bigint
- `pg_stat_get_db_tuples_inserted(oid)`  
Description: Specifies the number of tuples inserted in a database.  
Return type: bigint
- `pg_stat_get_db_tuples_updated(oid)`  
Description: Specifies the number of tuples updated in a database.  
Return type: bigint
- `pg_stat_get_db_tuples_deleted(oid)`  
Description: Specifies the number of tuples deleted in a database.  
Return type: bigint
- `pg_stat_get_db_conflict_lock(oid)`  
Description: Specifies the number of lock conflicts in a database.  
Return type: bigint

- `pg_stat_get_db_deadlocks(oid)`  
Description: Specifies the number of deadlocks in a database.  
Return type: `bigint`
- `pg_stat_get_numscans(oid)`  
Description: Number of sequential row scans done if parameters are in a table, or the number of index rows if parameters are in an index.  
Return type: `bigint`
- `pg_stat_get_role_name(oid)`  
Description: Obtains the username based on the user OID. Only users with the SYSADMIN and MONADMIN permissions can access the information.  
Return type: `text`  
Example:

```
gaussdb=# select pg_stat_get_role_name(10);
 pg_stat_get_role_name

 aabbcc
(1 row)
```
- `pg_stat_get_tuples_returned(oid)`  
Description: Specifies the number of sequential rows read by sequential scans if parameters are in a table, or the number of index rows if parameters are in an index.  
Return type: `bigint`
- `pg_stat_get_tuples_fetched(oid)`  
Description: Specifies the number of table rows fetched by bitmap scans if parameters are in a table, or the number of table rows fetched by simple index scans if parameters are in an index.  
Return type: `bigint`
- `pg_stat_get_tuples_inserted(oid)`  
Description: Specifies the number of rows inserted into a table.  
Return type: `bigint`
- `pg_stat_get_tuples_updated(oid)`  
Description: Specifies the number of rows updated in a table.  
Return type: `bigint`
- `pg_stat_get_tuples_deleted(oid)`  
Description: Specifies the number of rows deleted from a table.  
Return type: `bigint`
- `pg_stat_get_tuples_changed(oid)`  
Description: Specifies the total number of inserted, updated, and deleted rows after a table was last analyzed or autoanalyzed.  
Return type: `bigint`
- `pg_stat_get_tuples_hot_updated(oid)`  
Description: Specifies the number of rows hot updated in a table.  
Return type: `bigint`
- `pg_stat_get_live_tuples(oid)`

- Description: Specifies the number of live rows in a table.  
Return type: bigint
- `pg_stat_get_dead_tuples(oid)`

Description: Specifies the number of dead rows in a table.  
Return type: bigint
- `pg_stat_get_blocks_fetched(oid)`

Description: Specifies the number of disk block fetch requests for a table or an index.  
Return type: bigint
- `pg_stat_get_blocks_hit(oid)`

Description: Specifies the number of disk block requests found in cache for a table or an index.  
Return type: bigint
- `pg_stat_get_xact_tuples_fetched(oid)`

Description: Specifies the number of tuple rows scanned in a transaction.  
Return type: bigint
- `pg_stat_get_xact_tuples_inserted(oid)`

Description: Specifies the number of tuple inserted into the active subtransactions related to a table.  
Return type: bigint
- `pg_stat_get_xact_tuples_deleted(oid)`

Description: Specifies the number of deleted tuples in the active subtransactions related to a table.  
Return type: bigint
- `pg_stat_get_xact_tuples_hot_updated(oid)`

Description: Specifies the number of hot updated tuples in the active subtransactions related to a table.  
Return type: bigint
- `pg_stat_get_xact_tuples_updated(oid)`

Description: Specifies the number of updated tuples in the active subtransactions related to a table.  
Return type: bigint
- `pg_stat_get_last_vacuum_time(oid)`

Description: Specifies the most recent time when the autovacuum thread is manually started to clear a table.  
Return type: timestampz
- `pg_stat_get_last_autovacuum_time(oid)`

Description: Specifies the time of the last vacuum initiated by the autovacuum daemon on a table.  
Return type: timestampz
- `pg_stat_get_vacuum_count(oid)`

Description: Specifies the number of times a table is manually cleared.  
Return type: bigint

- `pg_stat_get_autovacuum_count(oid)`  
Description: Specifies the number of times the autovacuum daemon is started to clear a table.  
Return type: `bigint`
- `pg_stat_get_last_analyze_time(oid)`  
Description: Specifies the last time when a table starts to be analyzed manually or by the autovacuum thread.  
Return type: `timestampz`
- `pg_stat_get_last_autoanalyze_time(oid)`  
Description: Specifies the time when the last analysis initiated by the autovacuum daemon on a table.  
Return type: `timestampz`
- `pg_stat_get_analyze_count(oid)`  
Description: Specifies the number of times a table is manually analyzed.  
Return type: `bigint`
- `pg_stat_get_autoanalyze_count(oid)`  
Description: Specifies the number of times the autovacuum daemon analyzes a table.  
Return type: `bigint`
- `pg_total_autovac_tuples(bool)`  
Description: Returns tuple records related to the total autovac, such as **nodename**, **nspname**, **relname**, and tuple IUDs. The input parameter specifies whether to query the **relation** information.  
Return type: `setofrecord`  
The following table describes return parameters.

**Table 7-107** Return parameter description

| Return Parameter                   | Type                | Description                                             |
|------------------------------------|---------------------|---------------------------------------------------------|
| <code>nodename</code>              | <code>name</code>   | Node name.                                              |
| <code>nspname</code>               | <code>name</code>   | Name of a namespace                                     |
| <code>relname</code>               | <code>name</code>   | Name of an object, such as a table, an index, or a view |
| <code>partname</code>              | <code>name</code>   | Partition name                                          |
| <code>n_dead_tuples</code>         | <code>bigint</code> | Number of dead rows in a table partition                |
| <code>n_live_tuples</code>         | <code>bigint</code> | Number of live rows in a table partition                |
| <code>changes_since_analyze</code> | <code>bigint</code> | Number of changes generated by ANALYZE                  |

- pg\_total\_gsi\_autovac\_tuples(bool)**  
 Description: Returns tuple records related to **total autovac**, such as **nodename**, **nspname**, **relname**, and number of GSI records that have changed since the last ANALYZE. The input parameter specifies whether to query relation information. This function assists AUTOVACUUM on GSIs. You are advised not to use this function.  
 Return type: setofrecord
- pg\_autovac\_status(oid)**  
 Description: Returns autovac information, such as **nodename**, **nspname**, **relname**, **analyze**, **vacuum**, thresholds for the ANALYZE and VACUUM operations, and the number of analyzed or vacuumed tuples. Only users with the SYSADMIN permission can use this function.  
 Return type: setofrecord  
 The following table describes return parameters.

**Table 7-108** Return parameter description

| Return Parameter | Type    | Description                                              |
|------------------|---------|----------------------------------------------------------|
| nspname          | text    | Name of a namespace.                                     |
| relname          | text    | Name of an object, such as a table, an index, or a view. |
| nodename         | text    | Node name.                                               |
| doanalyze        | Boolean | Specifies whether to execute <b>ANALYZE</b> .            |
| anltuples        | bigint  | Number of ANALYZE tuples.                                |
| anlthresh        | bigint  | ANALYZE threshold.                                       |
| dovacuum         | Boolean | Specifies whether to execute <b>VACUUM</b> .             |
| vactuples        | bigint  | Number of VACUUM tuples.                                 |
| vacthresh        | bigint  | VACUUM threshold.                                        |

- pg\_autovac\_timeout(oid)**  
 Description: Returns the number of consecutive timeouts during the autovac operation on a table. If the table information is invalid or the node information is abnormal, **NULL** will be returned.  
 Return type: bigint
- pg\_autovac\_coordinator(oid)**  
 Description: Returns the name of the CN performing the autovac operation on a table. If the table information is invalid or the node information is abnormal, **NULL** is returned.



**Table 7-109** Return parameter description

| Return Parameter | Type                     | Description                                                                                                                                                                                                           |
|------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| datid            | oid                      | OID of the database that the user session connects to in the backend.                                                                                                                                                 |
| pid              | bigint                   | Backend thread ID.                                                                                                                                                                                                    |
| sessionid        | bigint                   | Session ID.                                                                                                                                                                                                           |
| usesysid         | oid                      | OID of the user logged in to the backend.                                                                                                                                                                             |
| application_name | text                     | Name of the application connected to the backend                                                                                                                                                                      |
| state            | text                     | Overall status of the backend                                                                                                                                                                                         |
| query            | text                     | Latest query at the backend. If <b>state</b> is <b>active</b> , this column shows the ongoing query. In all other states, it shows the last query that was executed.                                                  |
| waiting          | Boolean                  | Specifies whether the backend is currently waiting on a lock. If the backend is currently waiting, the value is <b>true</b> .                                                                                         |
| xact_start       | timestamp with time zone | Time when current transaction was started (null if no transaction is active).<br>If the current query is the first of its transaction, the value of this column is the same as that of the <b>query_start</b> column. |

| Return Parameter | Type                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| query_start      | timestamp with time zone | Time when the currently active query was started, or time when the last query was started if <b>state</b> is not <b>active</b> . For a stored procedure, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure. For a stored procedure or function, the first query time is queried and does not change with the running of statements in the stored procedure. |
| backend_start    | timestamp with time zone | Time when this process was started, that is, when the client connected to the server                                                                                                                                                                                                                                                                                                                                                       |
| state_change     | timestamp with time zone | Time when <b>state</b> was last modified                                                                                                                                                                                                                                                                                                                                                                                                   |
| client_addr      | inet                     | IP address of the client connected to the backend. If this column is <b>NULL</b> , it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as <b>AUTOVACUUM</b> .                                                                                                                                                                                                                 |
| client_hostname  | text                     | Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.                                                                                                                                                                                                                                          |

| Return Parameter | Type    | Description                                                                                            |
|------------------|---------|--------------------------------------------------------------------------------------------------------|
| client_port      | integer | TCP port number that the client uses for communication with this backend (-1 if a Unix socket is used) |
| enqueue          | text    | Unsupported currently.                                                                                 |
| query_id         | bigint  | ID of a query statement.                                                                               |
| srespool         | name    | Name of the resource pool                                                                              |
| global_sessionid | text    | Global session ID                                                                                      |
| unique_sql_id    | bigint  | Unique SQL statement ID                                                                                |
| trace_id         | text    | Driver-specific trace ID, which is associated with an application request                              |

- pg\_stat\_get\_activity\_with\_conninfo(integer)

Description: Returns a record about the backend with the specified PID. A record for each active backend in the system is returned if **NULL** is specified. The initial user, system administrators and users with the MONADMIN permission can view all data. Common users can only query their own results.

Return type: setofrecord

The following table describes return values.

**Table 7-110** Return value description

| Return Parameter | Return Type | Return value description                                              |
|------------------|-------------|-----------------------------------------------------------------------|
| datid            | oid         | OID of the database that the user session connects to in the backend. |
| pid              | bigint      | Backend thread ID.                                                    |
| sessionid        | bigint      | Session ID.                                                           |
| usesysid         | oid         | OID of the user logged in to the backend.                             |
| application_name | text        | Name of the application connected to the backend                      |

| Return Parameter | Return Type              | Return value description                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| state            | text                     | Overall status of the backend.                                                                                                                                                                                                                                                                                                                                                                                                             |
| query            | text                     | Latest query at the backend. If <b>state</b> is <b>active</b> , this column shows the ongoing query. In all other states, it shows the last query that was executed.                                                                                                                                                                                                                                                                       |
| waiting          | Boolean                  | Specifies whether the backend is currently waiting on a lock. If the backend is currently waiting, the value is <b>true</b> .                                                                                                                                                                                                                                                                                                              |
| xact_start       | timestamp with time zone | Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the <b>query_start</b> column.                                                                                                                                                                                                                         |
| query_start      | timestamp with time zone | Time when the currently active query was started, or time when the last query was started if <b>state</b> is not <b>active</b> . For a stored procedure, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure. For a stored procedure or function, the first query time is queried and does not change with the running of statements in the stored procedure. |

| Return Parameter | Return Type              | Return value description                                                                                                                                                                                                |
|------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| backend_start    | timestamp with time zone | Time when this process was started, that is, when the client connected to the server                                                                                                                                    |
| state_change     | timestamp with time zone | Time when <b>state</b> was last modified                                                                                                                                                                                |
| client_addr      | inet                     | IP address of the client connected to the backend. If this column is <b>NULL</b> , it indicates that either the client is connected via a Unix socket on the server or this is an internal process, such as AUTOVACUUM. |
| client_hostname  | text                     | Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.                       |
| client_port      | integer                  | TCP port number that the client uses for communication with this backend (-1 if a Unix socket is used)                                                                                                                  |
| enqueue          | text                     | Unsupported currently.                                                                                                                                                                                                  |
| query_id         | bigint                   | ID of a query statement.                                                                                                                                                                                                |
| connection_info  | text                     | A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database.                                                                                 |
| srespool         | name                     | Name of the resource pool                                                                                                                                                                                               |

| Return Parameter | Return Type | Return value description                                                  |
|------------------|-------------|---------------------------------------------------------------------------|
| global_sessionid | text        | Global session ID                                                         |
| unique_sql_id    | bigint      | Unique SQL statement ID                                                   |
| trace_id         | text        | Driver-specific trace ID, which is associated with an application request |
| top_xid          | xid         | Top-level transaction ID of a transaction.                                |
| current_xid      | xid         | Current transaction ID of a transaction.                                  |
| xlog_quantity    | bigint      | Amount of Xlogs currently used by a transaction, in bytes.                |

- `gs_get_explain(integer)`

Description: Returns a running plan for the background thread with the specified PID. The PID cannot be empty. This function takes effect only when the GUC parameter **track\_activities** is set to **on**. Only explainable SQL statements whose plans do not contain stream operators are supported. Details are as follows:

- If the GUC parameter **plan\_collect\_thresh** is set to **-1**, the return result of the function is always empty.
- If **plan\_collect\_thresh** is set to **0**, the current SQL execution time is greater than or equal to the value of **log\_min\_duration\_statement**, and the total number of tuples processed by all operators in the plan is greater than or equal to 10000, the system starts to collect plans in running state. Each time the total number of tuples processed by all operators exceeds 10000, a collection is performed.
- If the GUC parameter **plan\_collect\_thresh** is set to a value greater than 0, running plans are collected incrementally based on the threshold specified by this parameter.
- In addition, this function can view only the plans generated on CNs. To view the plans generated on DNs, see the `gs_get_dn_explain` function.

The return value type is text. The types and meanings of the fields are as follows:

| Return Value                                       | Return Type | Return value description                                                                             |
|----------------------------------------------------|-------------|------------------------------------------------------------------------------------------------------|
| Character string of the plans in the running state | text        | <b>A-rows</b> in the plan string indicates the number of rows returned by the operator in real time. |

- gs\_get\_dn\_explain(text, bigint)**  
 Description: Returns a running plan for the background thread of the DN with the specified node name (from the **node\_name** field in the **pgxc\_node** system catalog) and global session ID. The global session ID cannot be empty. This function takes effect only when the GUC parameter **track\_activities** is set to **on**. Only explainable SQL statements whose plans do not contain stream operators are supported. Details are as follows:
  - Same as the **gs\_get\_explain** function. The global session ID is calculated based on the **global\_sessionid** field in **pg\_stat\_activity**. The format of **global\_sessionid** is  $\${number1}:\${number2}\#\${number3}$ , for example, 1938253334:1#0. Therefore, the parameter value of the **gs\_get\_dn\_explain** function is  $\${number2} \times 10000000000 + \${number1}$ .
  - In addition, this function can be executed only with the ADMIN or MONADMIN permission.

The return value type is text. The types and meanings of the fields are as follows:

| Return Value                                       | Return Type | Return value description                                                                             |
|----------------------------------------------------|-------------|------------------------------------------------------------------------------------------------------|
| Character string of the plans in the running state | text        | <b>A-rows</b> in the plan string indicates the number of rows returned by the operator in real time. |

- pg\_stat\_get\_activity\_ng(integer)**  
 Description: Returns a record about the active background thread with the specified PID. A record for each active background thread is returned if **NULL** is specified. System administrators and users with the MONADMIN permission can view all data. Common users can query only their own data.

Return type: setofrecord

The following table describes return fields.

| Name      | Type   | Description       |
|-----------|--------|-------------------|
| datid     | oid    | Database OID      |
| pid       | bigint | Backend thread ID |
| sessionid | bigint | Session ID        |

- pg\_stat\_get\_function\_calls(oid)**  
 Description: Specifies the number of times the function has been called.  
 Return type: bigint
- pg\_stat\_get\_function\_self\_time(oid)**  
 Description: Specifies the time spent on only this function. The time spent on nested functions to call other functions is excluded.  
 Return type: bigint

- `pg_stat_get_backend_idset()`  
Description: Sets the number of currently active server processes (from 1 to the number of active server processes).  
Return type: setofinteger
- `pg_stat_get_backend_pid(integer)`  
Description: Specifies the ID of the given server thread.  
Return type: bigint
- `pg_stat_get_backend_dbid(integer)`  
Description: Specifies the ID of the database connected to the given server process.  
Return type: oid
- `pg_stat_get_backend_userid(integer)`  
Description: Specifies the user ID of the given server process. This function can be called only by system administrators.  
Return type: oid
- `pg_stat_get_backend_activity(integer)`  
Description: Active command of the given server process, but only if the current user is a system administrator or the same user as that of the session being queried and **track\_activities** is on  
Return type: text
- `pg_stat_get_backend_waiting(integer)`  
Description: True if the given server process is waiting for a lock, but only if the current user is a system administrator or the same user as that of the session being queried and **track\_activities** is on  
Return type: Boolean
- `pg_stat_get_backend_activity_start(integer)`  
Description: Specifies the time when the given server process's currently executing query is started only if the current user is a system administrator or the user of the session being queried and **track\_activities** is enabled.  
Return type: timestamp with time zone
- `pg_stat_get_backend_xact_start(integer)`  
Description: Specifies the time when the given server process's currently executing transaction is started only if the current user is a system administrator or the user of the session being queried and **track\_activities** is enabled.  
Return type: timestamp with time zone
- `pg_stat_get_backend_start(integer)`  
Description: Specifies the time when the given server process is started. If the current user is neither a system administrator nor the user of the session being queried, NULL is **returned**.  
Return type: timestamp with time zone
- `pg_stat_get_backend_client_addr(integer)`  
Description: Specifies the IP address of the client connected to the given server process. If the connection is over a UDS, or if the current user is neither

a system administrator nor the same user as that of the session being queried, **NULL** will be returned.

Return type: inet

- `pg_stat_get_backend_client_port(integer)`  
Description: Specifies the TCP port number of the client connected to the given server process. If the connection is over a UDS, **-1** will be returned. If the current user is neither a system administrator nor the same user as that of the session being queried, **NULL** will be returned.  
Return type: integer
- `pg_stat_get_bgwriter_timed_checkpoints()`  
Description: Specifies the time when the background writer starts scheduled checkpoints (because the **checkpoint\_timeout** time has expired).  
Return type: bigint
- `pg_stat_get_bgwriter_requested_checkpoints()`  
Description: Specifies the time when the background writer starts checkpoints based on requests from the backend because **checkpoint\_segments** has been exceeded or the **CHECKPOINT** command has been executed.  
Return type: bigint
- `pg_stat_get_bgwriter_buf_written_checkpoints()`  
Description: Specifies the number of buffers written by the background writer during checkpoints.  
Return type: bigint
- `pg_stat_get_bgwriter_buf_written_clean()`  
Description: Specifies the number of buffers written by the background writer for routine cleaning of dirty pages.  
Return type: bigint
- `pg_stat_get_bgwriter_maxwritten_clean()`  
Description: Specifies the time when the background writer stops its cleaning scan because it has written more buffers than specified in the **bgwriter\_lru\_maxpages** parameter.  
Return type: bigint
- `pg_stat_get_buf_written_backend()`  
Description: Specifies the number of buffers written by the backend because they need to allocate a new buffer.  
Return type: bigint
- `pg_stat_get_buf_alloc()`  
Description: Specifies the total number of the allocated buffers.  
Return type: bigint
- `pg_stat_clear_snapshot()`  
Description: Discards the current statistics snapshot. Only users with the SYSADMIN or MONADMIN permission can execute this function.  
Return type: void
- `pg_stat_reset()`

Description: Resets all statistics counters for the current database to zero (requires system administrator permissions).

Return type: void

- `gs_stat_reset()`

Description: Resets all statistics counters for the current database on each node to zero (requiring system administrator permissions).

Return type: void

- `pg_stat_reset_shared(text)`

Description: Resets all statistics counters for the current database in each node in a shared cluster to zero (requires system administrator permissions).

Return type: void

- `pg_stat_reset_single_table_counters(oid)`

Description: Resets statistics for a single table or index in the current database to zero (requires system administrator permissions).

Return type: void

- `pg_stat_reset_single_function_counters(oid)`

Description: Resets statistics for a single function in the current database to zero (requires system administrator permissions).

Return type: void

- `pgxc_get_wlm_current_instance_info(text, int default null)`

Description: Queries the current resource usage of each node in a cluster on a CN and reads the data that is not stored in the

**GS\_WLM\_INSTANCE\_HISTORY** system catalog in the memory. The input parameters are the node name (**ALL**, **C**, **D**, or *Instance name*) and the maximum number of records returned by each node. The returned value is **GS\_WLM\_INSTANCE\_HISTORY**.

Return type: setofrecord

- `pgxc_get_wlm_history_instance_info(text, TIMESTAMP, TIMESTAMP, int default null)`

Description: Queries the historical resource usage in a cluster on a CN and reads data from the **GS\_WLM\_INSTANCE\_HISTORY** system catalog. The input parameters are the node name (**ALL**, **C**, **D**, or *Instance name*), start time, end time, and maximum number of records returned by each instance. The return value is **GS\_WLM\_INSTANCE\_HISTORY**.

Return type: setofrecord

- `pgxc_fenced_udf_process(integer)`

Description: Displays the number of UDF master and worker processes. Only users with the SYSADMIN or MONADMIN permission can execute this function. If the input parameter is set to **1**, the number of Master processes is queried. If the input parameter is set to **2**, the number of Worker processes is queried. If the input parameter is set to **3**, all Worker processes are killed.

Return type: text

- `fenced_udf_process(integer)`

Description: Shows the number of local UDF Master and Work threads. If the input parameter is set to **1**, the number of Master threads is queried. If the

input parameter is set to **2**, the number of Worker threads is queried. If the input parameter is set to **3**, all Worker threads are terminated.

Return type: text

- total\_cpu()

Description: Obtains the CPU time used by the current node, in jiffies.

Return type: bigint

- total\_memory()

Description: Obtains the size of the virtual memory used by the current node, in KB.

Return type: bigint

- pgxc\_terminate\_all\_fenced\_udf\_process()

Description: Kills all UDF worker processes. Only users with the SYSADMIN or MONADMIN permission can execute this function.

Return type: Boolean

- GS\_ALL\_NODEGROUP\_CONTROL\_GROUP\_INFO(text)

Description: Provides Cgroup information for all node groups. Before calling this function, you need to specify the name of the logical cluster to be queried. For example, to query the Cgroup information for the **'installation'** logical cluster, run the following command:

```
SELECT * FROM GS_ALL_NODEGROUP_CONTROL_GROUP_INFO('installation')
```

Return type: record

The following table describes return columns.

| Name     | Type   | Description                                             |
|----------|--------|---------------------------------------------------------|
| name     | text   | Cgroup name.                                            |
| type     | text   | Cgroup type.                                            |
| gid      | bigint | Cgroup ID.                                              |
| classgid | bigint | ID of the class Cgroup where a workload Cgroup belongs. |
| class    | text   | Class Cgroup.                                           |
| workload | text   | Workload Cgroup.                                        |
| shares   | bigint | CPU quota allocated to the Cgroup.                      |
| limits   | bigint | Limit of CPU resources allocated to a Cgroup.           |
| wdlevel  | bigint | Workload Cgroup level.                                  |
| cpucores | text   | Information about the CPU cores used by a Cgroup.       |

- gs\_get\_nodegroup\_tablecount(name)

Description: Obtains the total number of user tables in all databases in a node group.

Return type: integer

- `pgxc_max_datanode_size(name)`  
Description: Obtains the maximum disk space occupied by database files on all DNs in a node group. The unit is byte.  
Return type: bigint
- `gs_check_tables_distribution()`  
Description: Checks whether the user table distribution in the system is consistent. If no record is returned, table distribution is consistent. This function cannot be called during redistribution in scale-in or scale-out.  
Return type: record
- `pg_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)`  
Description: Obtains damage information about pages after the current node is started.  
Return type: record
- `pgxc_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)`  
Description: Obtains damage information about pages after all nodes in the cluster are started.  
Return type: record
- `pg_stat_bad_block_clear()`  
Description: Deletes the page damage information that is read and recorded on the node (requires system administrator permissions).  
Return type: void
- `pgxc_stat_bad_block_clear`  
Description: Deletes the page damage information that is read and recorded on all nodes in the cluster (requiring system administrator permissions).  
Return type: void
- `gs_respool_exception_info(pool text)`  
Description: Queries the query rule of a specified resource pool.  
Return type: record
- `gs_control_group_info(pool text)`  
Description: Queries information about Cgroups associated with a resource pool. Only users with the SYSADMIN permission can execute this function.  
Return type: record

The command output is as follows:

| Attribute | Value               | Description                  |
|-----------|---------------------|------------------------------|
| name      | class_a:workload_a1 | Class name and workload name |
| class     | class_a             | Class Cgroup name            |
| workload  | workload_a1         | Workload Cgroup name         |

| Attribute | Value | Description                                                                              |
|-----------|-------|------------------------------------------------------------------------------------------|
| type      | DEFWD | Cgroup type ( <b>Top</b> , <b>CLASS</b> , <b>BAKWD</b> , <b>DEFWD</b> , or <b>TSWD</b> ) |
| gid       | 87    | Cgroup ID                                                                                |
| shares    | 30    | Percentage of CPU resources to those on the parent node                                  |
| limits    | 0     | Percentage of CPU cores to those on the parent node                                      |
| rate      | 0     | Allocation ratio in Timeshare                                                            |
| cpucores  | 0-3   | Number of CPU cores                                                                      |

- gs\_all\_control\_group\_info()**  
 Description: Collects information about all Cgroups in the database. For details about the columns returned by the function, see [16.3.48 GS\\_ALL\\_CONTROL\\_GROUP\\_INFO](#).  
 Return type: record
- gs\_get\_control\_group\_info()**  
 Description: Collects information about all Cgroups. For details about the columns returned by the function, see [16.3.53 GS\\_GET\\_CONTROL\\_GROUP\\_INFO](#). Only users with the SYSADMIN permission can execute this function.  
 Return type: record
- get\_instr\_workload\_info(integer)**  
 Description: Obtains the transaction volume and time information on the current CN.  
 Return type: record

| Attribute        | Value  | Description                                                    |
|------------------|--------|----------------------------------------------------------------|
| user_oid         | 10     | User ID                                                        |
| commit_counter   | 4      | Number of frontend transactions that were committed            |
| rollback_counter | 1      | Number of frontend transactions that were rolled back          |
| resp_min         | 949    | Minimum response time of frontend transactions (unit: $\mu$ s) |
| resp_max         | 201891 | Maximum response time of frontend transactions (unit: $\mu$ s) |
| resp_avg         | 43564  | Average response time of frontend transactions (unit: $\mu$ s) |

| Attribute           | Value        | Description                                                             |
|---------------------|--------------|-------------------------------------------------------------------------|
| resp_total          | 217822       | Total response time of frontend transactions (unit: $\mu\text{s}$ )     |
| bg_commit_counter   | 910          | Number of background transactions that were committed                   |
| bg_rollback_counter | 0            | Number of background transactions that were rolled back                 |
| bg_resp_min         | 97           | Minimum response time of background transactions (unit: $\mu\text{s}$ ) |
| bg_resp_max         | 678080687    | Maximum response time of background transactions (unit: $\mu\text{s}$ ) |
| bg_resp_avg         | 327847884    | Average response time of background transactions (unit: $\mu\text{s}$ ) |
| bg_resp_total       | 298341575300 | Total response time of background transactions (unit: $\mu\text{s}$ )   |

- `pv_instance_time()`  
Description: Obtains the time consumed in each execution phase on the current node.

Return type: record

| Stat_name Attribute | Value   | Description                                                                 |
|---------------------|---------|-----------------------------------------------------------------------------|
| DB_TIME             | 1062385 | Total end-to-end wall time consumed by all threads (unit: $\mu\text{s}$ )   |
| CPU_TIME            | 311777  | Total CPU time consumed by all threads (unit: $\mu\text{s}$ )               |
| EXECUTION_TIME      | 380037  | Total time consumed on the executor (unit: $\mu\text{s}$ )                  |
| PARSE_TIME          | 6033    | Total time consumed for parsing SQL statements (unit: $\mu\text{s}$ )       |
| PLAN_TIME           | 173356  | Total time consumed for generating an execution plan (unit: $\mu\text{s}$ ) |
| REWRITE_TIME        | 2274    | Total time consumed for rewriting queries (unit: $\mu\text{s}$ )            |

| Stat_name Attribute | Value  | Description                                                      |
|---------------------|--------|------------------------------------------------------------------|
| PL_EXECUTION_TIME   | 0      | Total time consumed for executing PL/SQL statements (unit: μs)   |
| PL_COMPILATION_TIME | 557    | Total time consumed for compiling SQL statements (unit: μs)      |
| NET_SEND_TIME       | 1673   | Total time consumed for sending data over the network (unit: μs) |
| DATA_IO_TIME        | 426622 | Total time consumed for reading and writing data (unit: μs)      |

- DBE\_PERF.get\_global\_instance\_time()**  
 Description: Provides the time consumed in each key phase in the entire cluster. The time consumed can be queried only on the CN. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
 Return type: record
- get\_instr\_unique\_sql()**  
 Description: Obtains information about execution statements (normalized SQL statements) on the current node. Only users with the SYSADMIN or MONADMIN permission can query this function.  
 Return type: record
- get\_instr\_wait\_event(integer)**  
 Description: Obtains the statistics on wait events of the current node.  
 Return type: record
- get\_instr\_user\_login()**  
 Description: Obtains the number of user login and logout times on the current node. Only users with the SYSADMIN or MONADMIN permission can execute this function.  
 Return type: record
- get\_instr\_rt\_percentile(integer)**  
 Description: Obtains the response time distribution for 80% and 95% SQL statements in the CCN. The unified cluster information is stored on the CCN. The query result from other nodes is 0.  
 Return type: record
- get\_node\_stat\_reset\_time()**  
 Description: Obtains statistics about reset (restart, primary/standby switchover, and database deletion) time of the current node.  
 Return type: record
- create\_wlm\_operator\_info(int flag)**  
 Description: Clears top SQL operator-level statistics recorded in the current memory. If the input parameter is greater than 0, the information is archived to gs\_wlm\_operator\_info. Otherwise, the information is not archived. Only users with the SYSADMIN permission can execute this function.

Return type: int

- create\_wlm\_session\_info(int flag)

Description: Clears top SQL query statement-level statistics recorded in the current memory. If the input parameter is greater than 0, the information is archived to **gs\_wlm\_session\_query\_info\_all**. Otherwise, the information is not archived. Only users with the SYSADMIN permission can execute this function.

Return type: int

- pg\_stat\_get\_wlm\_session\_info(int flag)

Description: Obtains top SQL query statement-level statistics recorded in the current memory. If the input parameter is not 0, the information is cleared from the memory. Only users with the SYSADMIN and MONADMIN permission can execute this function.

Return type: record

- gs\_paxos\_stat\_replication()

Description: Queries the standby node information on the primary node.

Return type: setofrecord

The following table describes return columns.

| Column                   | Type | Description                                                               |
|--------------------------|------|---------------------------------------------------------------------------|
| local_role               | text | Role of the node that sends logs                                          |
| peer_role                | text | Role of the node that receives logs                                       |
| local_dcf_role           | text | DCF role of the node that sends logs                                      |
| peer_dcf_role            | text | DCF role of the node that receives logs                                   |
| peer_state               | text | Status of the node that receives logs                                     |
| sender_write_location    | text | Location in the Xlog buffer where the node that sends logs is written     |
| sender_commit_location   | text | Consistency point reached for the DCF logs of the node that sends logs    |
| sender_flush_location    | text | Location in the Xlog disk where the node that sends logs is written       |
| sender_replay_location   | text | Location where the node that sends logs replays logs                      |
| receiver_write_location  | text | Location in the Xlog buffer where the node that receives logs is written  |
| receiver_commit_location | text | Consistency point reached for the DCF logs of the node that receives logs |
| receiver_flush_location  | text | Location in the Xlog disk where the node that receives logs is written    |
| receiver_replay_location | text | Location where the node that receives logs replays Xlogs                  |

| Column       | Type | Description                |
|--------------|------|----------------------------|
| sync_percent | text | Synchronization percentage |
| dcf_run_mode | int4 | DCF synchronization mode   |
| channel      | text | Channel information        |

- `get_paxos_replication_info()`  
Description: Queries the primary/standby replication status in Paxos mode.  
Return type: setofrecord

The following table describes return columns.

| Column                | Type | Description                                                                            |
|-----------------------|------|----------------------------------------------------------------------------------------|
| paxos_write_location  | text | Location of the Xlog that has been written to the Distribute Consensus Framework (DCF) |
| paxos_commit_location | text | Location of the Xlog agreed in the DCF                                                 |
| local_write_location  | text | Writing position of a node                                                             |
| local_flush_location  | text | Flushing position of a node                                                            |
| local_replay_location | text | Redo position of a node                                                                |
| dcf_replication_info  | text | DCF module information of a node                                                       |

- `gs_wlm_get_resource_pool_info(int)`  
Description: Obtains the resource usage statistics of all users. The input parameter is of the int type and can be any int value or **NULL**.  
Return type: record
- `gs_wlm_get_all_user_resource_info()`  
Description: Obtains resource usage statistics of all users. Only users with the SYSADMIN permission can execute this function.  
Return type: record
- `gs_wlm_get_user_info(int)`  
Description: Obtains information about all users. The input parameter is of the int type and can be any int value or **NULL**. Only users with the SYSADMIN permission can execute this function.  
Return type: record
- `gs_wlm_persistent_user_resource_info()`

Description: Archives all user resource usage statistics to the `gs_wlm_user_resource_history` system catalog. Only users with the `SYSADMIN` permission can execute this function.

Return type: record

- `gs_wlm_readjust_user_space(oid)`

Description: Corrects the storage space usage of all users. Only the administrator can execute this function.

Return type: record

- `gs_wlm_readjust_user_space_through_username(text name)`

Description: Corrects the storage space usage of a specified user. Common users can use this function to modify only their own usage. Only the administrator can modify the usage of all users. If the value of **name** is **0000**, the usage of all users needs to be modified.

Return type: record

- `gs_wlm_readjust_user_space_with_reset_flag(text name, boolean isfirst)`

Description: Corrects the storage space usage of a specified user. If the input parameter **isfirst** is set to **true**, statistics are collected from 0. Otherwise, statistics are collected from the previous result. Common users can use this function to modify only their own usage. Only the administrator can modify the usage of all users. If the value of **name** is **0000**, the usage of all users needs to be modified.

Return type: record

- `gs_wlm_session_respool(bigint)`

Description: Obtains the session resource pool information about all background threads. The input parameter is of the `bigint` type and can be set to any `bigint` value or **NULL**.

Return type: record

- `gs_total_nodegroup_memory_detail`

Description: Returns statistics on memory usage of the node group in the current database, in the unit of MB.

#### NOTE

If the GUC parameter **enable\_memory\_limit** is set to **off**, this function cannot be used.

Return type: SETOF record

**Table 7-111** Return value description

| Name   | Type | Description      |
|--------|------|------------------|
| ngname | text | Node group name. |

| Name        | Type    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| memorytype  | text    | Memory type. The value must be one of the following: <ul style="list-style-type: none"> <li>• <b>ng_total_memory</b>: total memory size of the node group.</li> <li>• <b>ng_used_memory</b>: memory used by the node group.</li> <li>• <b>ng_estimate_memory</b>: estimated memory used by the node group.</li> <li>• <b>ng_foreignrp_memsize</b>: total memory of the external resource pool of the node group.</li> <li>• <b>ng_foreignrp_usesize</b>: memory used by the external resource pool of the node group.</li> <li>• <b>ng_foreignrp_peaksize</b>: peak memory used by the external resource pool of the node group.</li> <li>• <b>ng_foreignrp_mempct</b>: percentage of the external resource pool of the node group to the total memory of the node group.</li> <li>• <b>ng_foreignrp_estmsize</b>: estimated memory used by the external resource pool of the node group.</li> </ul> |
| memorybytes | integer | Size of allocated memory-typed memory                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

- `gs_io_wait_status()`

Description: Returns the real-time statistics on I/O control on the current node.

Return type: SETOF record

| Name             | Type    | Description                                                                          |
|------------------|---------|--------------------------------------------------------------------------------------|
| node_name        | text    | Node name                                                                            |
| device_name      | text    | Name of the data disk mounted to the node.                                           |
| read_per_second  | float   | Number of read completions per second.                                               |
| write_per_second | float   | Number of write completions per second.                                              |
| write_ratio      | float   | Ratio of the disk write I/Os to the total I/Os.                                      |
| io_util          | float   | Percentage of the I/O time to the total CPU time per second.                         |
| total_io_util    | integer | Level of the CPU time occupied by the last three I/Os. The value ranges from 0 to 6. |

| Name                | Type    | Description                                                                                                                     |
|---------------------|---------|---------------------------------------------------------------------------------------------------------------------------------|
| tick_count          | integer | Interval for updating disk I/O information. The value is fixed to 1 second. The value is cleared each time before data is read. |
| io_wait_list_length | integer | Size of the I/O request thread wait queue. If the value is 0, no I/O is under control.                                          |

- `gs_get_shared_memctx_detail(text)`

Description: Returns the memory allocation details of the specified memory context, including the file, line number, and size of each memory allocation (the size of the same line in the same file is accumulated). Only the memory context queried through the `pg_shared_memory_detail` view can be queried. The input parameter is the memory context name (that is, the **contextname** column in the result returned by the `pg_shared_memory_detail` view). To query this function, you must have the SYSADMIN or MONADMIN permission.

Return type: SETOF record

| Name | Type | Description                                                                                                                             |
|------|------|-----------------------------------------------------------------------------------------------------------------------------------------|
| file | text | Name of the file where the memory is applied for.                                                                                       |
| line | int8 | Line number of the code in the file where the requested memory is located.                                                              |
| size | int8 | Size of the allocated memory. The value is accumulated if the memory is allocated for multiple times to the same line in the same file. |

#### NOTE

This view is not supported in the Lite release version.

- `gs_get_session_memctx_detail(text)`

Description: Returns the memory allocation details of the specified memory context, including the file, line number, and size of each memory allocation (the size of the same line in the same file is accumulated). This parameter is valid only in thread pool mode. Only the memory context queried through the `pv_session_memory_context` view can be queried. The input parameter is the memory context name (that is, the **contextname** column in the result returned by the `pv_session_memory_context` view). To query this function, you must have the SYSADMIN or MONADMIN permission.

Return type: SETOF record

| Name | Type | Description                                       |
|------|------|---------------------------------------------------|
| file | text | Name of the file where the memory is applied for. |

| Name | Type | Description                                                                                                                                       |
|------|------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| line | int8 | Line number of the code in the file where the requested memory is located.                                                                        |
| size | int8 | Size of the allocated memory, in bytes. The value is accumulated if the memory is allocated for multiple times to the same line of the same file. |

 **NOTE**

This view takes effect only in thread pool mode and is not supported in the Lite release version.

- `gs_get_thread_memctx_detail(tid,text)`

Description: Returns the memory allocation details of the specified memory context, including the file, line number, and size of each memory allocation (the size of the same line in the same file is accumulated). Only the memory context queried through the `pv_thread_memory_context` view can be queried. The first input parameter is the thread ID (the **tid** column of the data returned by `pv_thread_memory_context`), and the second parameter is the memory context name (the **contextname** column of the data returned by `pv_thread_memory_context`). To query this function, you must have the SYSADMIN or MONADMIN permission.

Return type: SETOF record

| Name | Type | Description                                                                                                                                       |
|------|------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| file | text | Name of the file where the memory is applied for.                                                                                                 |
| line | int8 | Line number of the code in the file where the requested memory is located.                                                                        |
| size | int8 | Size of the allocated memory, in bytes. The value is accumulated if the memory is allocated for multiple times to the same line of the same file. |

 **NOTE**

This view is not supported in the Lite release version.

- `gs_get_history_memory_detail(cstring)`

Description: Queries historical memory snapshot information. The input parameter type is `cstring`. The value can be **NULL** or the name of the memory snapshot log file.

- If the value of the input parameter is **NULL**, the list of all memory snapshot log files on the current node is displayed.
- If the value of the input parameter is the name of the memory snapshot log file in the list queried in [a](#), the detailed information about the memory snapshot recorded in the log file is displayed.

- c. If you enter any other input parameter, the system displays a message indicating that the input parameter is incorrect or the file fails to be opened.

To query this function, you must have the SYSADMIN or MONADMIN permission.

Return type: text

| Name        | Type | Description                                                                                                                                                                                                                                 |
|-------------|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| memory_info | text | Memory information. If the input parameter of the function is set to <b>NULL</b> , the memory snapshot file list is displayed. If the input parameter is set to the name of the memory snapshot file, the content of the file is displayed. |

- gs\_stat\_get\_hotkeys\_info()

 **NOTE**

If the GUC parameter **enable\_hotkeys\_collection** is set to **off**, the **gs\_stat\_get\_hotkeys\_info** and **global\_stat\_get\_hotkeys\_info** functions as well as the **global\_stat\_hotkeys\_info** view cannot be queried. The use of the **gs\_stat\_clean\_hotkeys** and **global\_stat\_clean\_hotkeys** APIs is not affected.

Description: Obtains the hotspot key statistics on the current node.

Return type: record

```
gaussdb=# select * from gs_stat_get_hotkeys_info() order by count, hash_value;
database_name | schema_name | table_name | key_value | hash_value | count
-----+-----+-----+-----+-----+-----
regression | public | hotkey_single_col | {22} | 1858004829 | 2
regression | public | hotkey_single_col | {11} | 2011968649 | 2
(2 rows)
```

Table 1 Return value description

| Name          | Type   | Description                                                                                                                              |
|---------------|--------|------------------------------------------------------------------------------------------------------------------------------------------|
| database_name | text   | Name of the database where the hotspot key is located                                                                                    |
| schema_name   | text   | Name of the schema where the hotspot key is located                                                                                      |
| table_name    | text   | Name of the table where the hotspot key is located                                                                                       |
| key_value     | text   | Value of the hotspot key                                                                                                                 |
| hash_value    | bigint | Hash value of the hotspot key in the database. If the table is a list or range distributed table, the value of this column is <b>0</b> . |
| count         | bigint | Frequency of accessing the hotspot key                                                                                                   |

- gs\_stat\_clean\_hotkeys()

 **NOTE**

- Hot key detection is designed for high-concurrency and heavy-traffic scenarios. In the scenario where the access is performed for several times, the query result may be inaccurate.
- The clearing API is designed to clear only the statistics in the LRU queue but not the historical data in the FIFO. Therefore, if the historical key value in the FIFO is accessed again after the clearing, the historical key value is still processed as a hotspot key. This rule also applies to **global\_stat\_clean\_hotkeys**.

Description: Clears statistics on hotspot keys on the current node.

Return type: Boolean

```
gaussdb=# select * from gs_stat_clean_hotkeys();
gs_stat_clean_hotkeys

t
(1 row)
```

- **global\_stat\_get\_hotkeys\_info()**

 **NOTE**

Run the **select \* from global\_stat\_hotkeys\_info minus select \* from global\_stat\_get\_hotkeys\_info()** command during service execution. The value may not be **0** due to time difference.

Description: Obtains statistics on hotspot keys in the entire cluster.

Return type: record

```
gaussdb=# select * from global_stat_get_hotkeys_info() order by count, hash_value;
database_name | schema_name | table_name | key_value | hash_value | count
-----+-----+-----+-----+-----+-----
regression | public | hotkey_single_col | {22} | 1858004829 | 2
regression | public | hotkey_single_col | {11} | 2011968649 | 2
(2 rows)
```

- **global\_stat\_clean\_hotkeys()**

Description: Clears statistics on hotspot keys in the entire cluster.

Return type: Boolean

```
gaussdb=# select * from global_stat_clean_hotkeys();
global_stat_clean_hotkeys

t
(1 row)
```

- **global\_comm\_get\_rcv\_stream()**

Description: Obtains the status of the stream received by all communications libraries on all DNs. For details about the columns returned by the function, see [PG\\_COMM\\_RECV\\_STREAM](#).

Return type: record

- **global\_comm\_get\_send\_stream()**

Description: Obtains the status of the stream sent by all communications libraries on all DNs. For details about the columns returned by the function, see [PG\\_COMM\\_SEND\\_STREAM](#).

Return type: record

- **global\_comm\_get\_status()**

Description: Obtains the communications library status on all DNs. For details about the columns returned by the function, see [PG\\_COMM\\_STATUS](#).

Return type: record

- global\_comm\_client\_info()**

Description: Obtains information about active client connections of global nodes. For details about the columns returned by the function, see [COMM\\_CLIENT\\_INFO](#).

Return type: record
- global\_comm\_get\_client\_info()**

Description: Obtains information about client connections of global nodes. For details about the columns returned by the function, see [COMM\\_CLIENT\\_INFO](#).

Return type: record
- pgxc\_get\_wlm\_operator\_history()**

Description: Displays the operator information when the execution of jobs cached on all CNs is complete. The information is cleared every 3 minutes. Only users with the SYSADMIN permission can execute this function.

Return type: record
- pgxc\_get\_wlm\_operator\_info()**

Description: Displays the operator information when the execution of jobs on all CNs is complete. Only users with the SYSADMIN permission can execute this function.

Return type: record
- pgxc\_get\_wlm\_operator\_statistics()**

Description: Displays the operator information when jobs on all CNs are being executed. Only users with the SYSADMIN permission can execute this function.

Return type: record
- pgxc\_stat\_activity()**

Description: Displays information about all CNs in the current cluster queried by the current user. Only users with the SYSADMIN or MONADMIN permission can execute this function, and common users can view only their own information.

Return type: record

| Name      | Type   | Description                                                           |
|-----------|--------|-----------------------------------------------------------------------|
| coorname  | text   | Name of a CN in the current cluster                                   |
| datid     | oid    | OID of the database that the user session connects to in the backend  |
| datname   | text   | Name of the database that the user session connects to in the backend |
| pid       | bigint | Backend thread ID                                                     |
| sessionid | bigint | Session ID                                                            |
| usesysid  | oid    | OID of the user logged in to the backend                              |

| Name             | Type                     | Description                                                                                                                                                                                                                |
|------------------|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| username         | text                     | Name of the user logged in to the backend                                                                                                                                                                                  |
| application_name | text                     | Name of the application connected to the backend                                                                                                                                                                           |
| client_addr      | inet                     | IP address of the client connected to the backend. If this column is <b>NULL</b> , it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as <b>AUTOVACUUM</b> . |
| client_hostname  | text                     | Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.                          |
| client_port      | integer                  | TCP port number that the client uses for communication with this backend (-1 if a Unix socket is used)                                                                                                                     |
| backend_start    | timestamp with time zone | Time when this process was started, that is, when the client connected to the server                                                                                                                                       |
| xact_start       | timestamp with time zone | Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the <b>query_start</b> column.         |
| query_start      | timestamp with time zone | Time when the currently active query was started, or time when the last query was started if <b>state</b> is not <b>active</b>                                                                                             |
| state_change     | timestamp with time zone | Time when <b>state</b> was last modified                                                                                                                                                                                   |
| waiting          | Boolean                  | Specifies whether the backend is currently waiting for a lock. If the backend is currently waiting for a lock, the value is <b>true</b> .                                                                                  |
| enqueue          | text                     | Queuing status of a statement. Its value can be: <ul style="list-style-type: none"><li>• <b>waiting in queue</b>: The statement is in the queue.</li><li>• <b>Empty</b>: The statement is running.</li></ul>               |

| Name          | Type   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| state         | text   | <p>Overall status of the backend Its value can be:</p> <ul style="list-style-type: none"> <li>• <b>active</b>: The backend is executing a query.</li> <li>• <b>idle</b>: The backend is waiting for a new client command.</li> <li>• <b>idle in transaction</b>: The backend is in a transaction, but there is no statement being executed in the transaction.</li> <li>• <b>idle in transaction (aborted)</b>: The backend is in a transaction, but there are statements failed in the transaction.</li> <li>• <b>fastpath function call</b>: The backend is executing a fast-path function.</li> <li>• <b>disabled</b>: This state is reported if <b>track_activities</b> is disabled in this backend.</li> </ul> <p><b>NOTE</b><br/>Only system administrators can view the session status of their accounts. The state information of other accounts is empty. For example, after user <b>judy</b> is connected to the database, the state information of user <b>joe</b> and the initial user <b>omm</b> in <b>pgxc_stat_activity</b> is empty.</p> <pre>SELECT datname, username, usesysid, state,pid FROM pgxc_stat_activity;  datname   username   usesysid   state    pid -----+-----+-----+-----+-----  postgres   omm             10           139968752121616  postgres   omm             10           139968903116560  db_tpcds   judy        16398   active    139968391403280  postgres   omm             10           139968643069712  postgres   omm             10           139968680818448  postgres   joe         16390           139968563377936 (6 rows)</pre> |
| resource_pool | name   | Resource pool used by the user                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| query_id      | bigint | ID of a query                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

| Name             | Type   | Description                                                                                                                                                          |
|------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| query            | text   | Latest query at the backend. If <b>state</b> is <b>active</b> , this column shows the ongoing query. In all other states, it shows the last query that was executed. |
| global_sessionid | text   | Global session ID                                                                                                                                                    |
| unique_sql_id    | bigint | Unique SQL statement ID                                                                                                                                              |
| trace_id         | text   | Driver-specific trace ID, which is associated with an application request                                                                                            |

- pgxc\_stat\_activity\_with\_conninfo()**

Description: Displays query information about the current user on all CNs in the current cluster. For details, see the **pgxc\_stat\_activity** view. Only users with the SYSADMIN or MONADMIN permission can execute this function, and common users can view only their own information.

Return type: record
- pgxc\_stat\_all\_tables()**

Description: Displays statistics on a row in each table (including TOAST tables) on each node. Only users with the SYSADMIN or MONADMIN permission can execute this function.

Return type: record
- pgxc\_get\_thread\_wait\_status()**

Description: Queries the call hierarchy between threads generated by all SQL statements on each node in a cluster and the blocking status of each thread.

Return type: record
- pgxc\_wlm\_get\_workload\_records()**

Description: Displays the status information when jobs on all CNs are being executed. Only system administrators can execute this function.

Return type: record
- pv\_session\_memory**

Description: Collects statistics on memory usage at the session level, including all the memory allocated to Postgres and stream threads on DN for jobs currently executed by users.

 **NOTE**

If the GUC parameter **enable\_memory\_limit** is set to **off**, this function cannot be used.

Return type: record

**Table 7-112** Return value description

| Name     | Type    | Description                                                                           |
|----------|---------|---------------------------------------------------------------------------------------|
| sessid   | text    | Thread start time and ID                                                              |
| init_mem | integer | Memory allocated to the currently executed jobs before they enter the executor, in MB |
| used_mem | integer | Memory allocated to the currently executed jobs, in MB                                |
| peak_mem | integer | Peak memory allocated to the currently executed jobs, in MB                           |

- DBE\_PERF.gs\_stat\_activity\_timeout(int)

Description: Obtains information about query jobs whose execution time exceeds the timeout threshold on the current node. The correct result can be returned only when the GUC parameter **track\_activities** is set to **on**. The timeout threshold ranges from 0 to 2147483. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

Return type: SETOF record

| Name             | Type       | Description                                               |
|------------------|------------|-----------------------------------------------------------|
| database         | name       | Name of the database to which a user session is connected |
| pid              | bigint     | Backend thread ID                                         |
| sessionid        | bigint     | Session ID                                                |
| usesysid         | oid        | OID of the user logged in to the backend                  |
| application_name | text       | Name of the application connected to the backend          |
| query            | text       | Query that is being executed on the backend               |
| xact_start       | timestampz | Time when the current transaction is started              |
| query_id         | bigint     | Query statement ID                                        |

- DBE\_PERF.global\_stat\_activity\_timeout(int)

Description: Obtains information about query jobs whose execution time exceeds the timeout threshold in the current system (all CNs). The correct result can be returned only when the GUC parameter **track\_activities** is set to **on**. The timeout threshold ranges from 0 to 2147483. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

Return type: SETOF record

| Name             | Type       | Description                                               |
|------------------|------------|-----------------------------------------------------------|
| nodename         | text       | Name of the CN connected to the user session              |
| database         | name       | Name of the database to which a user session is connected |
| pid              | bigint     | Backend thread ID                                         |
| sessionid        | bigint     | Session ID                                                |
| usesysid         | oid        | OID of the user logged in to the backend                  |
| application_name | text       | Name of the application connected to the backend          |
| query            | text       | Query that is being executed on the backend               |
| xact_start       | timestampz | Time when the current transaction is started              |
| query_start      | timestampz | Time when the current query starts                        |
| query_id         | bigint     | Query statement ID.                                       |

- `DBE_PERF.get_global_active_session()`  
 Description: Displays a summary of samples in the ACTIVE SESSION PROFILE memory on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
 Return type: record
- `DBE_PERF.get_global_os_runtime()`  
 Description: Displays the running status of the current OS. This function can be queried only on CNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
 Return type: record
- `DBE_PERF.get_global_os_threads()`  
 Description: Provides thread status information on all normal nodes in the entire cluster. The information can be queried only on CNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
 Return type: record
- `DBE_PERF.get_global_os_threads()`  
 Description: Provides thread status information on all normal nodes in the entire cluster. The information can be queried only on CNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
 Return type: record
- `DBE_PERF.get_summary_workload_sql_count()`  
 Description: Provides the count information of SELECT, UPDATE, INSERT, DELETE, DDL, DML, and DCL in different loads in the entire cluster. After a

cluster is created, by default, you must have the MONADMIN permission to query this function.

Return type: record

- `DBE_PERF.get_summary_workload_sql_elapse_time()`  
Description: Provides SELECT, UPDATE, INSERT, DELETE, and response time information (TOTAL, AVG, MIN, and MAX) in different loads in the entire cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_workload_transaction()`  
Description: Obtains the transaction volume and time information on all nodes in the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_session_stat()`  
Description: Obtains the session status information on all nodes in the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record

 **NOTE**

The status information contains the following 14 items:

commit,rollback,sql,table\_scan,blocks\_fetched,physical\_read\_operation,shared\_blocks\_dirtied,local\_blocks\_dirtied,shared\_blocks\_read,local\_blocks\_read,blocks\_read\_time,blocks\_write\_time,sort\_imemory,sort\_idisk

- `DBE_PERF.get_global_session_time()`  
Description: Provides the time consumed in each key phase on each node in the entire cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_session_memory()`  
Description: Aggregates statistics on memory usage at the session level on each node in the unit of MB, including all the memory allocated to Postgres and stream threads on DNs for jobs currently executed by users. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_session_memory_detail()`  
Description: Aggregates statistics on thread memory usage on each node by the MemoryContext node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_session_stat_activity()`  
Description: Aggregates information about running threads on each node in the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

- Return type: record
- DBE\_PERF.get\_global\_thread\_wait\_status()  
Description: Aggregates the blocking waiting status of the backend thread and auxiliary thread on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_wlm\_controlgroup\_ng\_config()  
Description: Collects information about all Cgroups in the database. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_wlm\_workload\_runtime()  
Description: Aggregates the status information about jobs executed by the current user on each CN. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_operator\_history\_table()  
Description: Aggregates the operator records (persistent) after jobs are executed by the current user on all CNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_operator\_history()  
Description: Aggregates the operator records after jobs are executed by the current user on all CNs. After a cluster is created, by default, you must have the MONADMIN or SYSADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_operator\_runtime()  
Description: Aggregates real-time operator records of jobs executed by the current user on all CNs. After a cluster is created, by default, you must have the MONADMIN or SYSADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statement\_complex\_history()  
Description: Aggregates the historical records of complex queries executed by the current user on all CNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statement\_complex\_history\_table()  
Description: Aggregates the historical records of complex queries (persistent) executed by the current user on all CNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statement\_complex\_runtime()  
Description: Aggregates real-time information about complex queries executed by the current user on all CNs. After a cluster is created, by default, you must have the MONADMIN or SYSADMIN permission to query this function.

- Return type: record
- DBE\_PERF.get\_global\_memory\_node\_detail()  
Description: Aggregates the memory usage of a database on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_shared\_memory\_detail()  
Description: Aggregates the usage information about the shared memory contexts on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_comm\_delay()  
Description: Aggregates the communications library delay status on all DNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_comm\_recv\_stream()  
Description: Aggregates the status of the stream received by the communications library on all DNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_comm\_send\_stream()  
Description: Aggregates the status of the stream sent by the communications library on all DNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_comm\_status()  
Description: Aggregates the status of the communications library on all DNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statio\_all\_indexes()  
Description: Aggregates index information and I/O statistics in the current database on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_local\_toastname\_and\_toastindexname()  
Description: Provides the mapping between the name and index of the local TOAST table and its associated tables. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_statio\_all\_indexes()  
Description: Collects I/O statistics on specific indexes, covering all index lines in the current database on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

- Return type: record
- DBE\_PERF.get\_global\_statio\_all\_sequences()  
Description: Provides I/O status information about all sequences in the namespace. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statio\_all\_tables()  
Description: Aggregates I/O statistics on each table in the database on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_statio\_all\_tables()  
Description: Collects statistics on I/Os of each table in the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_local\_toast\_relation()  
Description: Provides the mapping between the name of the local TOAST table and its associated tables. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statio\_sys\_indexes()  
Description: Aggregates I/O status information about all system catalog indexes in namespaces on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_statio\_sys\_indexes()  
Description: Collects statistics on I/O status information about all system catalog indexes in namespaces on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statio\_sys\_sequences()  
Description: Provides I/O status information about all system catalog sequences in namespaces. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statio\_sys\_tables()  
Description: Provides I/O status information about all system catalogs in the namespaces on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_statio\_sys\_tables()  
Description: Aggregates I/O status information about all system catalogs in the namespaces of the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

- Return type: record
- DBE\_PERF.get\_global\_statio\_user\_indexes()  
Description: Provides I/O status information about all user relationship table indexes in the namespaces on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_statio\_user\_indexes()  
Description: Aggregates I/O status information about all user relationship table indexes in the namespaces of the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statio\_user\_sequences()  
Description: Provides I/O status information about all user sequences in the namespaces on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statio\_user\_tables()  
Description: Provides I/O status information about all user relationship tables in the namespaces on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_statio\_user\_tables()  
Description: Aggregates I/O status information about all user relationship tables in the namespaces of the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_dn\_stat\_all\_tables()  
Description: Aggregates statistics on all tables in the database on each DN. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_cn\_stat\_all\_tables()  
Description: Aggregates statistics on all tables in the database on each CN. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_dn\_stat\_all\_tables()  
Description: Collects statistics on all tables in the database on each DN. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_cn\_stat\_all\_tables()  
Description: Collects statistics on all tables in the database on each CN. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

- Return type: record
- DBE\_PERF.get\_global\_stat\_all\_indexes()  
Description: Aggregates statistics on all indexes in the database on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_stat\_all\_indexes()  
Description: Collects statistics on all indexes in the database on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_stat\_sys\_tables()  
Description: Aggregates statistics on the system catalogs of all the namespaces in the pg\_catalog or information\_schema schema on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_stat\_sys\_tables()  
Description: Collects statistics on the system catalogs of all the namespaces in the pg\_catalog or information\_schema schema on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_stat\_sys\_indexes()  
Description: Aggregates index status information about all system catalogs in the pg\_catalog or information\_schema schema on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_stat\_sys\_indexes()  
Description: Collects index status information about all system catalogs in the pg\_catalog or information\_schema schema on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_stat\_user\_tables()  
Description: Aggregates status information about user-defined ordinary tables in all namespaces. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_stat\_user\_tables()  
Description: Collects status information about user-defined ordinary tables in all namespaces. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record

- `DBE_PERF.get_global_stat_user_indexes()`  
Description: Aggregates status information about the indexes of user-defined ordinary tables in all databases. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_summary_stat_user_indexes()`  
Description: Collects status information about the indexes of user-defined ordinary tables in all databases. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_stat_database()`  
Description: Aggregates statistics on databases on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_stat_database_conflicts()`  
Description: Collects statistics on databases on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_stat_xact_all_tables()`  
Description: Aggregates transaction status information about all ordinary tables and TOAST tables in namespaces. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_summary_stat_xact_all_tables()`  
Description: Collects transaction status information about all ordinary tables and TOAST tables in the namespace. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_stat_xact_sys_tables()`  
Description: Aggregates transaction status information about system catalogs in namespaces on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_summary_stat_xact_sys_tables()`  
Description: Collects transaction status information about system catalogs in namespaces on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_stat_xact_user_tables()`  
Description: Aggregates transaction status information about user tables in namespaces on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record

- `DBE_PERF.get_summary_stat_xact_user_tables()`  
Description: Collects transaction status information about user tables in namespaces on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_stat_user_functions()`  
Description: Aggregates transaction status information about user-defined functions in namespaces on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_stat_xact_user_functions()`  
Description: Collects transaction status information about user-defined functions in namespaces on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_stat_bad_block()`  
Description: Aggregates information about the failure to read files such as tables and indexes on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_file_redo_iostat()`  
Description: Collects information about the failure to read files such as tables and indexes on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_file_iostat()`  
Description: Collects I/O statistics of data files on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_locks()`  
Description: Aggregates lock information on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_replication_slots()`  
Description: Aggregates logical replication information on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.GET_GLOBAL_PARALLEL_DECODE_STATUS()`  
Description: Displays the parallel decoding information of replication slots on all primary DNs in a cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function. This function can be executed only on CNs. An error is reported when it is executed on DNs. The

returned value is the same as that of the [GLOBAL\\_PARALLEL\\_DECODE\\_STATUS](#) view.

Return type: record

- `DBE_PERF.GET_GLOBAL_PARALLEL_DECODE_THREAD_INFO()`  
Description: Displays the parallel decoding thread information of replication slots on all primary DNs in a cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function. This function can be executed only on CNs. An error is reported when it is executed on DNs. The returned value is the same as that of the [GLOBAL\\_PARALLEL\\_DECODE\\_THREAD\\_INFO](#) view.  
Return type: record
- `DBE_PERF.get_global_bgwriter_stat()`  
Description: Aggregates statistics on the backend write process activities on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_replication_stat()`  
Description: Aggregates status information about log synchronization on all nodes, such as the location where the sender sends logs and the location where the receiver receives logs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_pooler_status()`  
Description: Aggregates cache connection status in the pooler on all CNs. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_transactions_running_xacts()`  
Description: Aggregates information about running transactions on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_summary_transactions_running_xacts()`  
Description: Collects information about running transactions on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_global_transactions_prepared_xacts()`  
Description: Aggregates information about transactions that are currently prepared for two-phase commit on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `DBE_PERF.get_summary_transactions_prepared_xacts()`  
Description: Collects information about transactions that are currently prepared for two-phase commit on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

- Return type: record
- DBE\_PERF.get\_summary\_statement()  
Description: Aggregates the status of historical statements executed on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_statement\_count()  
Description: Aggregates SELECT, UPDATE, INSERT, DELETE, and response time information (TOTAL, AVG, MIN, and MAX) on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_config\_settings()  
Description: Aggregates GUC parameter settings on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_wait\_events()  
Description: Aggregates status information about the wait events on each node. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_statement\_responsetime\_percentile()  
Description: Obtains the response time distribution for 80% and 95% SQL statements in the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_summary\_user\_login()  
Description: Collects statistics on the number of user login and logout times on each node in the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.get\_global\_record\_reset\_time()  
Description: Aggregates statistics on reset (restart, primary-standby switchover, and database deletion) time in the cluster. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
  - DBE\_PERF.standby\_statement\_history(only\_slow[, time1, time2])  
Description: Queries full SQL statements on the standby node. The primary node queries full SQL statements using the statement\_history table, while the standby node queries using this function. To query this function, you must have the MONADMIN permission.  
For details, see [Table 7-113](#).  
Return type: record, which is the same as that in statement\_history.

**Table 7-113** standby\_statement\_history parameters

| Parameter | Type       | Description                                                                                                                                                                                                                                                                                                                                                |
|-----------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| only_slow | Boolean    | Specifies whether to query only slow SQL statements. <ul style="list-style-type: none"> <li>• <b>true</b>: yes, which is equivalent to <code>select .. where is_slow_sql = true</code>.</li> <li>• <b>false</b> or <b>NULL</b> indicates that all SQL statements are queried, that is, <b>is_slow_sql</b> is not used as a filtering condition.</li> </ul> |
| time1     | timestampz | Minimum time specified by <b>finish_time</b> for querying SQL statements. This parameter is optional.                                                                                                                                                                                                                                                      |
| time2     | timestampz | Maximum time specified by <b>finish_time</b> for querying SQL statements. This parameter is optional.                                                                                                                                                                                                                                                      |

 **NOTE**

- The two time parameters **time1** and **time2** indicate the time segment to which **finish\_time** of the queried SQL statement belongs. They indicate the start time and end time respectively. If **NULL** or no value is entered, there is no limit. The function of **time1** and **time2** is the same as that of **select .. where finish\_time between time1 and time2**.
- The data generated from this function on the standby node is not stored in a table, and there is no index on the **start\_time** column. You are advised to use the parameter to search for **finish\_time**.
- Full/Slow SQL statements on the standby node are written to disks asynchronously. Therefore, the storage of user SQL information may be delayed. You are advised to query this API to expand the query time range.

- `DBE_PERF.track_memory_context(context_list text)`

Description: Sets the memory context whose memory allocation details need to be collected. The input parameter is the memory context names, which are separated by commas (,), for example, **ThreadTopMemoryContext**, **SessionCacheMemoryContext**. Note that the memory context names are context-sensitive. In addition, the length of a single memory context is 63, and the excess part is truncated. The maximum number of memory contexts that can be collected at a time is 16. If the number of memory contexts exceeds 16, the setting fails. Each time this function is called, the previous statistics result is cleared. When the input parameter is set to "", the statistics function is disabled. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

Return type: Boolean

- `DBE_PERF.track_memory_context_detail()`

Description: Obtains the memory allocation details of the memory context specified by the **DBE\_PERF.track\_memory\_context** function. For details, see the `DBE_PERF.track_memory_context_detail` view. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

Return type: record

- `DBE_PERF.global_io_wait_info()`  
Description: Queries real-time I/O control statistics on all CNs and DN. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `pg_stat_get_mem_mbytes_reserved(tid)`  
Description: Collects statistics on variables related to resource management, which is used only for fault locating.  
Parameter: thread ID  
Return type: text
- `gs_wlm_user_resource_info(name text)`  
Description: Queries a user's resource quota and resource usage. Common users can query only their own information. Administrators can query information about all users.  
Return type: record
- `pg_stat_get_file_stat()`  
Description: Records statistics about data file I/Os to indicate I/O performance and detect performance problems such as abnormal I/O operations.  
Return type: record
- `pg_stat_get_redo_stat()`  
Description: Displays statistics on the replay of session thread logs.  
Return type: record
- `pg_stat_get_status(int8)`  
Description: Tests the blocking status about the backend thread and auxiliary thread of the current instance.  
Return type: record
- `get_local_rel_iostat()`  
Description: Queries the accumulated I/O status of data files on the current node.  
Return type: record
- `DBE_PERF.get_global_rel_iostat()`  
Description: Collects I/O statistics of data files on all nodes. After a cluster is created, by default, you must have the MONADMIN permission to query this function.  
Return type: record
- `pg_catalog.plancache_status()`  
Description: Displays status information about the global plan cache on the current node. The information returned by the function is the same as that in [GLOBAL\\_PLANCACHE\\_STATUS](#).  
Return type: record
- `DBE_PERF.global_plancache_status()`  
Description: Displays status information about the global plan cache on all nodes. For details about the information returned by the function, see [GLOBAL\\_PLANCACHE\\_STATUS](#). After a cluster is created, by default, you must have the MONADMIN permission to query this function.

Return type: record

- Pg\_catalog.prepare\_statement\_status() (Discarded)

Description: Displays the PREPARE statement status information on the current node. The information returned by the function is the same as that in [GLOBAL\\_PREPARE\\_STATEMENT\\_STATUS](#).

Return type: record

- DBE\_PERF.global\_prepare\_statement\_status() (Discarded)

Description: Displays status information about the PREPARE statement on all nodes. For details about the information returned by the function, see [GLOBAL\\_PREPARE\\_STATEMENT\\_STATUS](#). After a cluster is created, by default, you must have the MONADMIN permission to query this function.

Return type: record

- DBE\_PERF.global\_threadpool\_status()

Description: Displays the status of worker threads and sessions in thread pools on all nodes. For details about the information returned by the function, see [18.7.14-Table GLOBAL\\_THREADPOOL\\_STATUS](#). After a cluster is created, by default, you must have the MONADMIN permission to query this function.

Return type: record

- comm\_check\_connection\_status

Description: Returns the connection status between the CN and all active nodes (CNs and primary DNs). This function can be queried only on CNs and can be used by common users.

Parameter: nan

Return type: node\_name text, remote\_name text, remote\_host text, remote\_port integer, is\_connected boolean, and no\_error\_occur boolean

- DBE\_PERF.global\_comm\_check\_connection\_status()

Description: Returns the connection status between all CNs and all active nodes (CNs and primary DNs). This function can be queried only on CNs. Permission control is inherited from the **DBE\_PERF** schema. After a cluster is created, by default, you must have the MONADMIN permission to query this function.

Parameter: nan

Return type: node\_name text, remote\_name text, remote\_host text, remote\_port integer, is\_connected boolean, and no\_error\_occur boolean

- create\_wlm\_instance\_statistics\_info

Description: Saves the historical monitoring data of the current instance persistently.

Parameter: nan

Return type: integer

- remote\_candidate\_stat()

Description: Displays the number of pages in the candidate buffer chain of this instance and buffer elimination information, including the normal buffer pool and segment buffer pool.

Return type: record

**Table 7-114** remote\_candidate\_stat parameter description

| Name                    | Type    | Description                                                                                                                           |
|-------------------------|---------|---------------------------------------------------------------------------------------------------------------------------------------|
| node_name               | text    | Node name                                                                                                                             |
| candidate_slots         | integer | Number of pages in the candidate buffer chain of the current normal buffer pool                                                       |
| get_buf_from_list       | bigint  | Number of times that pages are obtained from the candidate buffer chain during buffer eviction in the current normal buffer pool      |
| get_buf_clock_sweep     | bigint  | Number of times that pages are obtained from the original eviction solution during buffer eviction in the current normal buffer pool  |
| seg_candidate_slots     | integer | Number of pages in the candidate buffer chain of the current segment buffer pool                                                      |
| seg_get_buf_from_list   | bigint  | Number of times that pages are obtained from the candidate buffer chain during buffer eviction in the current segment buffer pool     |
| seg_get_buf_clock_sweep | bigint  | Number of times that pages are obtained from the original eviction solution during buffer eviction in the current segment buffer pool |

- remote\_ckpt\_stat()

Description: Displays the checkpoint information and log flushing information about all instances in the cluster (unavailable on DNs, except for the current node).

Return type: record

**Table 7-115** remote\_ckpt\_stat parameter description

| Parameter             | Type | Description                                                                |
|-----------------------|------|----------------------------------------------------------------------------|
| node_name             | text | Instance name                                                              |
| ckpt_redo_point       | text | Checkpoint of the current instance                                         |
| ckpt_clog_flush_num   | int8 | Number of Clog flushing pages from the startup time to the current time    |
| ckpt_csnlog_flush_num | int8 | Number of CSN log flushing pages from the startup time to the current time |

| Parameter                | Type | Description                                                                  |
|--------------------------|------|------------------------------------------------------------------------------|
| ckpt_multixact_flush_num | int8 | Number of MultiXact flushing pages from the startup time to the current time |
| ckpt_predicate_flush_num | int8 | Number of predicate flushing pages from the startup time to the current time |
| ckpt_twophase_flush_num  | int8 | Number of two-phase flushing pages from the startup time to the current time |

- remote\_double\_write\_stat()

Description: Displays doublewrite file status of all instances in the cluster (unavailable on DNs, except for the current node).

Return type: record

**Table 7-116** remote\_double\_write\_stat parameter description

| Parameter             | Type | Description                                                                                                                          |
|-----------------------|------|--------------------------------------------------------------------------------------------------------------------------------------|
| node_name             | text | Instance name                                                                                                                        |
| curr_dwn              | int8 | Sequence number of the doublewrite file.                                                                                             |
| curr_start_page       | int8 | Start page for restoring the doublewrite file                                                                                        |
| file_trunc_num        | int8 | Number of times that the doublewrite file is reused.                                                                                 |
| file_reset_num        | int8 | Number of reset times after the doublewrite file is full.                                                                            |
| total_writes          | int8 | Total number of I/Os of the doublewrite file                                                                                         |
| low_threshold_writes  | int8 | Number of I/Os for writing the doublewrite files with low efficiency (the number of I/O flushing pages at a time is less than 16.)   |
| high_threshold_writes | int8 | Number of I/Os for writing the doublewrite files with high efficiency (the number of I/O flushing pages at a time is more than 421.) |
| total_pages           | int8 | Total number of pages that are flushed to the doublewrite file area                                                                  |
| low_threshold_pages   | int8 | Number of pages that are flushed with low efficiency                                                                                 |
| high_threshold_pages  | int8 | Number of pages that are flushed with high efficiency                                                                                |

| Parameter | Type | Description                        |
|-----------|------|------------------------------------|
| file_id   | int8 | ID of the current doublewrite file |

- remote\_single\_flush\_dw\_stat()

Description: Displays the single-page doublewrite file eviction status of all instances in the cluster (unavailable on DNs, except for the current node).

Return type: record

**Table 7-117** remote\_single\_flush\_dw\_stat parameter description

| Parameter       | Type    | Description                                                       |
|-----------------|---------|-------------------------------------------------------------------|
| node_name       | text    | Instance name                                                     |
| curr_dwn        | integer | Sequence number of the doublewrite file.                          |
| curr_start_page | integer | Start position of the current doublewrite file.                   |
| total_writes    | bigint  | Total number of data write pages in the current doublewrite file. |
| file_trunc_num  | bigint  | Number of times that the doublewrite file is reused.              |
| file_reset_num  | bigint  | Number of reset times after the doublewrite file is full.         |

- remote\_pagewriter\_stat()

Description: Displays the page flushing information and checkpoint information about all instances in the cluster (unavailable on DNs, except for the current node).

Return type: record

**Table 7-118** remote\_pagewriter\_stat parameter description

| Parameter                   | Type | Description                                                                   |
|-----------------------------|------|-------------------------------------------------------------------------------|
| node_name                   | text | Instance name                                                                 |
| pgwr_actual_flush_total_num | int8 | Total number of dirty pages flushed from the startup time to the current time |
| pgwr_last_flush_num         | int4 | Number of dirty pages flushed in the previous batch                           |
| remain_dirty_page_num       | int8 | Estimated number of dirty pages that are not flushed                          |

| Parameter               | Type | Description                                                                                 |
|-------------------------|------|---------------------------------------------------------------------------------------------|
| queue_head_page_rec_lsn | text | <b>recovery_lsn</b> of the first dirty page in the dirty page queue of the current instance |
| queue_rec_lsn           | text | <b>recovery_lsn</b> of the dirty page queue of the current instance                         |
| current_xlog_insert_lsn | text | Write position of Xlogs in the current instance                                             |
| ckpt_redo_point         | text | Checkpoint of the current instance                                                          |

- remote\_recovery\_status()

Description: Displays log flow control information about the primary and standby nodes (except the current node and DNs).

Return type: record

**Table 7-119** remote\_recovery\_status parameter description

| Parameter          | Type | Description                                                                                           |
|--------------------|------|-------------------------------------------------------------------------------------------------------|
| node_name          | text | Node name (including the primary and standby nodes)                                                   |
| standby_node_name  | text | Name of the standby node                                                                              |
| source_ip          | text | IP address of the primary node                                                                        |
| source_port        | int4 | Port number of the primary node                                                                       |
| dest_ip            | text | IP address of the standby node                                                                        |
| dest_port          | int4 | Port number of the standby node                                                                       |
| current_rto        | int8 | Current log flow control time of the standby node (unit: s)                                           |
| target_rto         | int8 | Expected flow control time of the standby node specified by the corresponding GUC parameter (unit: s) |
| current_sleep_time | int8 | Sleep time required to achieve the expected flow control time (unit: $\mu$ s)                         |

- remote\_rto\_stat()

Description: Displays log flow control information about the primary and standby nodes (except the current node and DNs).

Return type: record

**Table 7-120** remote\_rto\_stat parameters

| Parameter | Type | Description                                                                                                                                                                                                                                                                |
|-----------|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| node_name | text | Node name (including the primary and standby nodes)                                                                                                                                                                                                                        |
| rto_info  | text | Flow control information, including the current log flow control time (unit: second) of the standby node, the expected flow control time (unit: second) specified by the GUC parameter, and the primary node sleep time (unit: $\mu$ s) required to reach the expectation. |

- remote\_redo\_stat()

Description: Displays the log replay status of all instances in the cluster (unavailable on DNs, except for the current node).

Return type: record

**Table 7-121** remote\_redo\_stat parameter description

| Parameter              | Type | Description                                                     |
|------------------------|------|-----------------------------------------------------------------|
| node_name              | text | Instance name                                                   |
| redo_start_ptr         | int8 | Start point for replaying the instance logs                     |
| redo_start_time        | int8 | Start time (UTC) when the instance logs are replayed            |
| redo_done_time         | int8 | End time (UTC) when the instance logs are replayed              |
| curr_time              | int8 | Current time (UTC) of the instance                              |
| min_recovery_point     | int8 | Position of the minimum consistency point for the instance logs |
| read_ptr               | int8 | Position for reading the instance logs                          |
| last_replayed_read_ptr | int8 | Position for replaying the instance logs                        |
| recovery_done_ptr      | int8 | Replay position after the instance is started                   |
| read_xlog_io_counter   | int8 | Number of I/Os when the instance reads and replays logs         |

| Parameter                 | Type | Description                                                                                                |
|---------------------------|------|------------------------------------------------------------------------------------------------------------|
| read_xlog_io_total_dur    | int8 | Total I/O latency when the instance reads and replays logs                                                 |
| read_data_io_counter      | int8 | Number of data page I/O reads during replay in the instance                                                |
| read_data_io_total_dur    | int8 | Total I/O latency of data page reads during replay in the instance                                         |
| write_data_io_counter     | int8 | Number of data page I/O writes during replay in the instance                                               |
| write_data_io_total_dur   | int8 | Total I/O latency of data page writes during replay in the instance                                        |
| process_pending_counter   | int8 | Number of synchronization times of log distribution threads during replay in the instance                  |
| process_pending_total_dur | int8 | Total synchronization latency of log distribution threads during replay in the instance                    |
| apply_counter             | int8 | Number of synchronization times of replay threads during replay in the instance                            |
| apply_total_dur           | int8 | Total synchronization latency of replay threads during replay in the instance                              |
| speed                     | int8 | Log replay rate of the current instance                                                                    |
| local_max_ptr             | int8 | Maximum number of replay logs received by the local host after the instance is started                     |
| primary_flush_ptr         | int8 | Log point where the host flushes logs to a disk                                                            |
| worker_info               | text | Replay thread information of the instance. If concurrent replay is not enabled, the value is <b>NULL</b> . |

- pgxc\_gtm\_snapshot\_status()

Description: Queries for transaction information on the current GTM. This function is supported only in GTM mode, and is not supported in GTM-LITE or GTM-FREE mode.

Return type: record

The following table describes return parameters.

**Table 7-122** PGXC\_GTM\_SNAPSHOT\_STATUS return parameters

| Name         | Type    | Description                                                                  |
|--------------|---------|------------------------------------------------------------------------------|
| xmin         | xid     | Minimum XID of the running transactions                                      |
| xmax         | xid     | XID of the transaction next to the executed transaction with the maximum XID |
| csn          | integer | Specifies the sequence number of the transaction to be committed.            |
| oldestxmin   | xid     | Minimum XID of the executed transactions                                     |
| xcnt         | integer | Number of the running transactions                                           |
| running_xids | text    | XID of the running transaction                                               |

- pv\_os\_run\_info()**

Description: Displays the running status of the current OS. For details about the columns, see [PV\\_OS\\_RUN\\_INFO](#).

Parameter: nan

Return type: SETOF record
- pv\_session\_stat()**

Description: Collects session status information by session thread or AutoVacuum thread. For details about the columns, see [PV\\_SESSION\\_STAT](#).

Parameter: nan

Return type: SETOF record
- pv\_session\_time()**

Description: Collects statistics on the running time of session threads and the time consumed in each execution phase. For details about the columns, see [PV\\_SESSION\\_TIME](#).

Parameter: nan

Return type: SETOF record
- pg\_stat\_get\_db\_temp\_bytes()**

Description: Collects statistics on the total amount of data written to temporary files through database query. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log\_temp\_files** setting.

Parameter: **oid**

- Return type: bigint
- pg\_stat\_get\_db\_temp\_files()**

Description: Queries the number of temporary files created in the database. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the **log\_temp\_files** setting.

Parameter: **oid**

Return type: bigint
  - gs\_prepared\_statements()**

Description: Displays prepared statements that are available in all sessions. Only users with the SYSADMIN permission can execute this function. The columns in the information returned by the function are the same as those in [GS\\_ALL\\_PREPARED\\_STATEMENTS](#).

Return type: record
  - local\_redo\_time\_count()**

Description: Returns the time consumption statistics on each process of each replayer thread on the current node (valid data exists only on the standby node).

The return values are as follows:

**Table 7-123** local\_redo\_time\_count return parameters

| Column      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| thread_name | Thread name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| step1_total | <p>Total duration of step 1. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li> <p>Ultimate RTO:</p> <p><b>redo batch</b>: obtains a log from a queue.</p> <p><b>redo manager</b>: obtains a log from a queue.</p> <p><b>redo worker</b>: obtains a log from a queue.</p> <p><b>txn manager</b>: reads a log from a queue.</p> <p><b>txn worker</b>: reads a log from a queue.</p> <p><b>read worker</b>: reads an Xlog page (overall) from a file.</p> <p><b>read page worker</b>: obtains a log from a queue.</p> <p><b>startup</b>: obtains a log from a queue.</p> </li> <li> <p>Parallel replay:</p> <p><b>page redo</b>: obtains a log from a queue.</p> <p><b>startup</b>: reads a log.</p> </li> </ul> |
| step1_count | Number of accumulated execution times of step 1.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

| Column      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| step2_total | <p>Total duration of step 2. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li>• Ultimate RTO: <ul style="list-style-type: none"> <li><b>redo batch</b>: processes logs (overall).</li> <li><b>redo manager</b>: processes logs (overall).</li> <li><b>redo worker</b>: processes logs (overall).</li> <li><b>txn manager</b>: processes logs (overall).</li> <li><b>txn worker</b>: processes logs (overall).</li> <li><b>read worker</b>: specifies the time required for reading the Xlog page.</li> <li><b>read page worker</b>: generates and sends LSN forwarders.</li> <li><b>startup</b>: checks whether to replay to the specified position.</li> </ul> </li> <li>• Parallel replay: <ul style="list-style-type: none"> <li><b>startup</b>: checks whether to replay to the specified position.</li> </ul> </li> </ul>                    |
| step2_count | Number of accumulated execution times of step 2.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| step3_total | <p>Total duration of step 3. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li>• Ultimate RTO: <ul style="list-style-type: none"> <li><b>redo batch</b>: updates the standby state.</li> <li><b>redo manager</b>: processes data logs.</li> <li><b>redo worker</b>: replays page logs (overall).</li> <li><b>txn manager</b>: updates the flushing LSN.</li> <li><b>txn worker</b>: replays logs.</li> <li><b>read worker</b>: pushes the Xlog segment.</li> <li><b>read page worker</b>: obtains a new item.</li> <li><b>startup</b>: collects statistics on the wait time of delayed replay feature.</li> </ul> </li> <li>• Parallel replay: <ul style="list-style-type: none"> <li><b>page redo</b>: updates the standby state.</li> <li><b>startup</b>: collects statistics on the wait time of delayed replay feature.</li> </ul> </li> </ul> |
| step3_count | Number of accumulated execution times of step 3.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

| Column      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| step4_total | <p>Total duration of step 4. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li>• Ultimate RTO: <ul style="list-style-type: none"> <li><b>redo batch</b>: parses Xlogs.</li> <li><b>redo manager</b>: processes DDL operations.</li> <li><b>redo worker</b>: reads data pages.</li> <li><b>txn manager</b>: synchronizes the wait time.</li> <li><b>txn worker</b>: updates the LSN of the current thread.</li> <li><b>read page worker</b>: stores logs in the distribution thread.</li> <li><b>startup</b>: distributes logs (overall).</li> </ul> </li> <li>• Parallel replay: <ul style="list-style-type: none"> <li><b>page redo</b>: replays undo logs.</li> <li><b>startup</b>: distributes logs (overall).</li> </ul> </li> </ul>               |
| step4_count | Number of accumulated execution times of step 4.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| step5_total | <p>Total duration of step 5. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li>• Ultimate RTO: <ul style="list-style-type: none"> <li><b>redo batch</b>: distributes to the redo manager.</li> <li><b>redo manager</b>: distributes logs to redo workers.</li> <li><b>redo worker</b>: replays data page logs.</li> <li><b>txn manager</b>: distributes data to the txn worker.</li> <li><b>txn worker</b>: forcibly synchronizes the wait time.</li> <li><b>read page worker</b>: updates the LSN of the current thread.</li> <li><b>startup</b>: decodes logs.</li> </ul> </li> <li>• Parallel replay: <ul style="list-style-type: none"> <li><b>page redo</b>: replays sharetxn logs.</li> <li><b>startup</b>: replays logs.</li> </ul> </li> </ul> |
| step5_count | Number of accumulated execution times of step 5.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

| Column      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| step6_total | <p>Total duration of step 6. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li>• Ultimate RTO:<br/><b>redo worker</b>: replays non-data page logs.<br/><b>trxn manager</b>: updates global LSNs.<br/><b>redo manager</b>: stores data page logs to the hash table.</li> <li>• Parallel replay:<br/><b>page redo</b>: replays synctrx logs.<br/><b>startup</b>: forcibly synchronizes the wait time.</li> </ul> |
| step6_count | Number of accumulated execution times of step 6.                                                                                                                                                                                                                                                                                                                                                                                                  |
| step7_total | <p>Total duration of step 7. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li>• Ultimate RTO:<br/><b>redo manager</b>: creates tablespaces.<br/><b>redo worker</b>: updates FSM.</li> <li>• Parallel replay:<br/><b>page redo</b>: replays a single log.</li> </ul>                                                                                                                                           |
| step7_count | Number of accumulated execution times of step 7.                                                                                                                                                                                                                                                                                                                                                                                                  |
| step8_total | <p>Total duration of step 8. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li>• Ultimate RTO:<br/><b>redo worker</b>: forcibly synchronizes the wait time.</li> <li>• Parallel replay:<br/><b>page redo</b>: replays all workers do logs.</li> </ul>                                                                                                                                                          |
| step8_count | Number of accumulated execution times of step 8.                                                                                                                                                                                                                                                                                                                                                                                                  |
| step9_total | <p>Total duration of step 9. The process of each thread is as follows:</p> <ul style="list-style-type: none"> <li>• Ultimate RTO:<br/><b>redo manager</b>: distributes logs to the page redo thread.</li> <li>• Parallel replay:<br/><b>page redo</b>: replays multi-worker do logs.</li> </ul>                                                                                                                                                   |
| step9_count | Number of accumulated execution times of step 9.                                                                                                                                                                                                                                                                                                                                                                                                  |

- `local_xlog_redo_statics()`  
Description: Returns the statistics on each type of logs that have been replayed on the current node (valid data exists only on the standby node).  
The return values are as follows:

**Table 7-124** `local_xlog_redo_statics` parameters

| Column                 | Description                                                                                                                                                                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>xlog_type</code> | Log types.                                                                                                                                                                                                                                               |
| <code>rmid</code>      | resource manager id                                                                                                                                                                                                                                      |
| <code>info</code>      | xlog operation                                                                                                                                                                                                                                           |
| <code>num</code>       | Number of logs.                                                                                                                                                                                                                                          |
| <code>extra</code>     | Valid values are available for page replay logs and xact logs. <ul style="list-style-type: none"> <li>• Number of pages read from the disk if the log is of the page type.</li> <li>• Number of deleted files if the log is of the xact type.</li> </ul> |

- `remote_bgwriter_stat()`  
Description: Displays the information about pages flushed by the bgwriter threads of all instances in the entire cluster, number of pages in the candidate buffer chain, and buffer elimination information. (The query result does not contain the information about the current node and cannot be used on DNs.)  
Return type: record

**Table 7-125** `remote_bgwriter_stat` parameter description

| Parameter                                | Type    | Description                                                                                          |
|------------------------------------------|---------|------------------------------------------------------------------------------------------------------|
| <code>node_name</code>                   | text    | Instance name                                                                                        |
| <code>bgwr_actual_flush_total_num</code> | bigint  | Total number of dirty pages flushed by the bgwriter thread from the startup time to the current time |
| <code>bgwr_last_flush_num</code>         | integer | Number of dirty pages flushed by the bgwriter thread in the previous batch                           |
| <code>candidate_slots</code>             | integer | Number of pages in the current candidate buffer chain                                                |
| <code>get_buffer_from_list</code>        | bigint  | Number of times that pages are obtained from the candidate buffer chain during buffer eviction       |

| Parameter           | Type   | Description                                                                                        |
|---------------------|--------|----------------------------------------------------------------------------------------------------|
| get_buf_clock_sweep | bigint | Number of times that pages are obtained from the original eviction solution during buffer eviction |

Example:

The remote\_bgwriter\_stat function is used to query the page refreshing information of the bgwriter thread.

```
gaussdb=# SELECT * FROM remote_bgwriter_stat();
 node_name | bgwr_actual_flush_total_num | bgwr_last_flush_num | candidate_slots |
get_buffer_from_list | get_buf_clock_sweep
-----+-----+-----+-----+-----+-----
 datanode3 | 0 | 0 | 266232 | 404 | 0
 datanode2 | 0 | 0 | 266232 | 424 | 0
 datanode1 | 0 | 0 | 266232 | 393 | 0
(3 rows)
```

- **gs\_stack()**

Description: Displays the call stack of a thread. To query this function, you must have the SYSADMIN or MONADMIN permission.

Parameter: tid, which indicates the thread ID. **tid** is an optional parameter. If it is specified, the function returns the call stack of the thread corresponding to **tid**. If it is not specified, the function returns the call stacks of all threads.

Return value: If **tid** is specified, the return value is of the TEXT type. If **tid** is not specified, the return value is a SETOF record.

Example:

Run **select \* from gs\_stack(tid)** to obtain the call stack of a specified thread.

```
gaussdb=# select * from gs_stack(139663481165568);
 gs_stack

__poll + 0x2d
WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f
WaitLatch(Latch volatile*, int, long) + 0x2e
JobScheduleMain() + 0x90f
int GaussDbThreadMain<(knl_thread_role)9>(knl_thread_arg*) + 0x456+
InternalThreadFunc(void*) + 0x2d
ThreadStarterFunc(void*) + 0xa4
start_thread + 0xc5
clone + 0x6d
(1 row)
```

Run **select \* from gs\_stack()** to obtain the call stacks of all threads.

```
gaussdb=# select * from gs_stack();
-[RECORD
1]-----
tid | 139670364324352
lwtid | 308
stack | __poll + 0x2d
 | | CommWaitPollParam::caller(int (*)(pollfd*, unsigned long, int), unsigned long) + 0x34
 | | int comm_socket_call<CommWaitPollParam, int (*)(pollfd*, unsigned long,
int)>(CommWaitPollParam*, int (*)(pollfd*, unsigned long
, int)) + 0x28
 | | comm_poll(pollfd*, unsigned long, int) + 0xb1
 | | ServerLoop() + 0x72b
 | | PostmasterMain(int, char**) + 0x314e
```

```

| main + 0x617
| __libc_start_main + 0xf5
| 0x55d38f8db3a7
[RECORD 2]-----
tid | 139664851859200
lwtid | 520
stack | __poll + 0x2d
 | WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f
 | SysLoggerMain(int) + 0xc86
 | int GaussDbThreadMain<(knl_thread_role)17>(knl_thread_arg*) + 0x45d
 | InternalThreadFunc(void*) + 0x2d
 | ThreadStarterFunc(void*) + 0xa4
 | start_thread + 0xc5
 | clone + 0x6d

```

- `gs_tpworker_execstmt_stat()`

Description: Displays the runtime information of a statement. If the SYSADMIN or MONADMIN user runs the statement, the information about all the statements that are being executed is displayed. Common users can query only the information about the SQL statements executed by themselves.

Return type: SETOF record

| Name                     | Type    | Description                                                                                                                                                                                       |
|--------------------------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| db_oid                   | oid     | OID of the database that the user session connects to in the backend                                                                                                                              |
| db_name                  | name    | Name of the database that the user session connects to in the backend                                                                                                                             |
| threadpool_worker        | varchar | NUMA group to which a thread belongs and thread ID. The format is <i>numagroup_threadid</i> .                                                                                                     |
| thread_id                | bigint  | Thread ID.                                                                                                                                                                                        |
| session_id               | bigint  | Session ID                                                                                                                                                                                        |
| query_id                 | bigint  | ID of the SQL statement that is being executed                                                                                                                                                    |
| query_text               | text    | Content of the SQL statement that is being executed                                                                                                                                               |
| unique_sql_id            | bigint  | Unique ID generated by the SQL statement                                                                                                                                                          |
| client_hostname          | text    | Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled. |
| client_app_name          | text    | Name of the client app                                                                                                                                                                            |
| stmt_slow_time_threshold | int     | Preset timeout interval for marking an SQL statement as a slow SQL statement, in milliseconds                                                                                                     |

| Name                 | Type    | Description                                                                                                                                                                                                                                                                                                                   |
|----------------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| stmt_elapse_time     | int     | Time elapsed since the query starts                                                                                                                                                                                                                                                                                           |
| stmt_control_status  | varchar | Current statement state <ul style="list-style-type: none"> <li>• <b>Waiting:</b> The session access is successful but it is not executed by the thread.</li> <li>• <b>Running:</b> The current statement is executed properly.</li> <li>• <b>Control:</b> The current statement enters the resource control phase.</li> </ul> |
| stmt_control_rule    | text    | Slow SQL control rule corresponding to the current language                                                                                                                                                                                                                                                                   |
| stmt_control_iostat  | text    | IOPS value and maximum IOPS of the current statement The format is as follows:<br>curVal/maxVal.                                                                                                                                                                                                                              |
| stmt_control_memstat | text    | This field is reserved and is not supported currently.                                                                                                                                                                                                                                                                        |
| stmt_control_cpustat | text    | This field is reserved and is not supported currently.                                                                                                                                                                                                                                                                        |
| stmt_control_netstat | text    | This field is reserved and is not supported currently.                                                                                                                                                                                                                                                                        |

- `gs_tpworker_execslot_stat()`  
Description: Displays the thread running information. If the SYSADMIN or MONADMIN user runs the command, information about all threads is displayed. Common users can query only information about the threads where the SQL statements executed by themselves are located.

Return type: SETOF record

| Name                | Type | Description                                                                                                  |
|---------------------|------|--------------------------------------------------------------------------------------------------------------|
| numagroup           | int  | NUMA group to which the current thread belongs                                                               |
| worker_id           | int  | Thread ID of the current thread                                                                              |
| worker_bind_type    | text | Thread binding mode. The value can be <b>numabind</b> , <b>cpubind</b> , <b>allbind</b> , or <b>nobind</b> . |
| worker_cpu_affinity | text | Affinity between threads and CPU cores, that is, the range of CPU cores that can be scheduled by threads.    |

| Name              | Type    | Description                                                                                                                                                                                                                                                                                                                  |
|-------------------|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| worker_status     | varchar | Current thread status: <ul style="list-style-type: none"> <li>• <b>Waiting:</b> The session access is successful but it is not executed by the thread.</li> <li>• <b>Running:</b> The current statement is executed properly.</li> <li>• <b>Control:</b> The current statement enters the resource control phase.</li> </ul> |
| served_query_id   | bigint  | ID of the SQL statement that is being executed                                                                                                                                                                                                                                                                               |
| served_query_text | text    | Content of the SQL statement that is being executed                                                                                                                                                                                                                                                                          |

- `gs_session_all_settings(sessionid bigint)`  
 Description: Queries the full GUC parameter settings of the session corresponding to the session ID on the local node. To execute this function, you must have the SYSADMIN or MONADMIN permission.  
 Input parameter description: **sessionid** indicates the session ID.  
 Return type: SETOF record  
 The following table describes return fields.

| Name    | Type | Description                  |
|---------|------|------------------------------|
| name    | text | Parameter name               |
| setting | text | Current parameter value      |
| unit    | text | Implicit unit of a parameter |

Example:

```
gaussdb=# select sessionid from pg_stat_activity where username = 'testuser';
sessionid

 788861
(1 row)

gaussdb=# select * from gs_session_all_settings(788861) where name = 'work_mem';
 name | setting | unit
-----+-----+-----
work_mem | 131072 | kB
(1 row)
```

- `gs_session_all_settings()`  
 Description: Queries full GUC parameter settings of all sessions on the local node. To execute this function, you must have the SYSADMIN or MONADMIN permission.  
 Return type: SETOF record

| Name      | Type   | Description                  |
|-----------|--------|------------------------------|
| sessionid | bigint | Session ID                   |
| pid       | bigint | Backend thread ID            |
| name      | text   | Parameter name               |
| setting   | text   | Current parameter value      |
| unit      | text   | Implicit unit of a parameter |

Example:

```
gaussdb=# select * from gs_session_all_settings() where name = 'work_mem';
 sessionid | pid | name | setting | unit
-----+-----+-----+-----+-----
140550214145792 | 96974 | work_mem | 65536 | kB
140550214145792 | 96971 | work_mem | 65536 | kB
140549731735296 | 140549731735296 | work_mem | 65536 | kB
140549764413184 | 140549764413184 | work_mem | 65536 | kB
(4 rows)
```

- `gs_local_wal_preparse_statistics()`

Description: Queries the latest startup of the log pre-parsing thread on the local node as well as the pre-parsing logs. Only the user with the SYSADMIN permission can execute this function.

Return type: SETOF record

| Name                     | Type        | Description                                                                       |
|--------------------------|-------------|-----------------------------------------------------------------------------------|
| preparser_term           | text        | Maximum term value obtained from the latest pre-parsing log.                      |
| preparser_start_time     | timestamptz | Time when the latest pre-parsing is started.                                      |
| preparser_end_time       | timestamptz | End time of the latest pre-parsing.                                               |
| preparser_start_location | text        | Start position of the latest pre-parsing log.                                     |
| preparser_end_location   | text        | End position of the latest pre-parsing log.                                       |
| preparser_total_bytes    | int8        | Number of latest pre-parsed logs, in bytes.                                       |
| preparser_speed          | int8        | Latest pre-parsing speed, in bytes/ms.                                            |
| is_valid                 | bool        | Specifies whether the latest pre-parsing result can be used for leader selection. |

Example:

```
gaussdb=# select * from gs_local_wal_preparse_statistics();
preparser_term | preparser_start_time | preparser_end_time | preparser_start_location |
preparser_end_location | preparser_total_bytes | preparser_speed | is_valid
-----+-----+-----+-----+-----+-----+-----+-----
3107 | 2023-02-01 17:04:23.367946+08 | 2023-02-01 17:04:25.354434+08 | 00000003/
C3EEA660 | 00000004/0BE60738 | 1207394520 | 1207394520 | f
(1 row)
```

- gs\_hot\_standby\_space\_info()

Description: Queries the total number and total size of files in the **standby\_read/base\_page**, **standby\_read/block\_info\_meta** and **standby\_read/lsn\_info\_meta** folders.

Return type: SETOF record

| Name                       | Type | Description                              |
|----------------------------|------|------------------------------------------|
| base_page_file_num         | xid  | Total number of bage_page_files.         |
| base_page_total_size       | xid  | Total size of bage_page_files.           |
| lsn_info_meta_file_num     | xid  | Total number of lsn_info_meta_files.     |
| lsn_info_meta_total_size   | xid  | Total size of the lsn_info_meta_files.   |
| block_info_meta_file_num   | xid  | Total number of block_info_meta_files.   |
| block_info_meta_total_size | xid  | Total size of the block_info_meta_files. |

Example:

```
gaussdb=# select * from gs_hot_standby_space_info();
base_page_file_num | base_page_total_size | lsn_info_meta_file_num | lsn_info_meta_total_size |
block_info_meta_file_num | block_info_meta_total_size
-----+-----+-----+-----+-----+-----
6 | 163840 | 6 | 3136 | 16
| 147456
(1 row)
```

- exrto\_file\_read\_stat()

Description: Queries the number of disk access times and total access latency of new **base page files**, **lsn info meta files**, and **block info meta files** read by the standby node. Connect to the standby DN for query. In other cases, the query result is 0.

Return type: SETOF record

| Name                            | Type | Description                                               |
|---------------------------------|------|-----------------------------------------------------------|
| lsn_info_page_disk_read_counter | int8 | Number of disk access times of <b>lsn info meta files</b> |

| Name                            | Type | Description                                                   |
|---------------------------------|------|---------------------------------------------------------------|
| lsn_info_page_disk_read_dur     | int8 | Total latency of <b>lsn info meta file</b> access to disks.   |
| blk_info_meta_disk_read_counter | int8 | Number of disk access times of <b>block info meta files</b>   |
| blk_info_meta_disk_read_dur     | int8 | Total latency of <b>block info meta file</b> access to disks. |
| base_page_read_disk_counter     | int8 | Number of disk access times of <b>base page files</b>         |
| base_page_read_disk_dur         | int8 | Total latency of <b>base page file</b> access to disks.       |

Example:

```
gaussdb=# SELECT * FROM exrto_file_read_stat();
lsn_info_page_disk_read_counter | lsn_info_page_disk_read_dur | blk_info_meta_disk_read_counter |
blk_info_meta_disk_read_dur | base_page_read_disk_counter | base_page_read_disk_dur
-----+-----+-----+-----+-----+-----
 14987 | 92313 | 23879 | 129811
| 0 | 0
(1 row)
```

- `gs_exrto_recycle_info()`

Description: Queries the resource reclamation location, including the reclamation LSN of each thread, global reclamation LSN, and the earliest snapshot LSN of a query thread. Connect to the standby DN for query. In other cases, the query result is 0.

Return type: SETOF record

| Name                       | Type | Description                                                                             |
|----------------------------|------|-----------------------------------------------------------------------------------------|
| page_redo_worker_thread_id | text | Reclamation LSN location of redo thread. <b>thread_id</b> indicates the redo thread ID. |
| global_recycle_lsn         | text | LSN of global reclamation location                                                      |
| exrto_snapshot_oldest_lsn  | text | The earliest snapshot LSN of a query thread                                             |

Example:

```
gaussdb=# SELECT * FROM gs_exrto_recycle_info();
thread_id | recycle_lsn
-----+-----
page_redo_worker_140148895381248 | 0/7B4552E0
page_redo_worker_140148872312576 | 0/7B4535B8
```

```
global_recycle_lsn | 0/7B4535B8
exrto_snapshot_oldest_lsn | 0/8488E6D0
(4 rows)
```

- gs\_stat\_get\_db\_conflict\_all(oid)

Input parameter: **dbid(oid)** indicates the database OID.

Description: Queries the number of sent replay conflict signals of different types.

Return type: SETOF record

| Name                           | Type | Description                                                                     |
|--------------------------------|------|---------------------------------------------------------------------------------|
| conflict_all                   | int8 | Number of sent replay conflict signals                                          |
| conflict_tablespace            | int8 | Number of sent replay conflict signals of the <b>tablespace</b> type            |
| conflict_lock                  | int8 | Number of sent replay conflict signals of the <b>lock</b> type                  |
| conflict_snapshot              | int8 | Number of sent replay conflict signals of the <b>snapshot</b> type              |
| conflict_bufferpin             | int8 | Number of sent replay conflict signals of the <b>bufferpin</b> type             |
| conflict_startup_deadlock      | int8 | Number of sent replay conflict signals of the <b>startup_deadlock</b> type      |
| conflict_truncate              | int8 | Number of sent replay conflict signals of the <b>truncate</b> type              |
| conflict_standby_query_timeout | int8 | Number of sent replay conflict signals of the <b>standby_query_timeout</b> type |
| conflict_force_recycle         | int8 | Number of sent replay conflict signals of the <b>force_recycle</b> type         |

Example:

```
gaussdb=# SELECT * FROM gs_stat_get_db_conflict_all(12738);
conflict_all | conflict_tablespace | conflict_lock | conflict_snapshot | conflict_bufferpin |
conflict_startup_deadlock | conflict_truncate | conflict_standby_query_timeout | conflict_force_recycle
-----+-----+-----+-----+-----+-----+-----+-----+-----
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
(1 row)
```

- gs\_redo\_stat\_info()

Description: Queries redo information, including the buffer hit ratio of the redo thread, number of unlink\_rels files executed, wait event information of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario, and wait event information of **wal\_read\_from\_write\_buffer**. The query must be executed by connecting to the standby DN.

Return type: SETOF record

| Name                     | Type   | Description                                                                                                                 |
|--------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------|
| buffer_hit_rate          | float8 | Buffer hit ratio of the redo thread.                                                                                        |
| ddl_unlink_nrels_count   | int8   | Number of unlink rel files executed during the redo process of DDL statements.                                              |
| read_buffer_io_counter   | int8   | Number of wait events of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario.       |
| read_buffer_io_total_dur | int8   | Total wait event duration of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario.   |
| read_buffer_io_avg_dur   | int8   | Average wait event duration of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario. |
| read_buffer_io_min_dur   | int8   | Minimum wait event duration of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario. |
| read_buffer_io_max_dur   | int8   | Maximum wait event duration of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario. |
| read_wal_buf_counter     | int8   | Number of wait events triggered by <b>wal_read_from_write_buffer</b> in the ultimate RTO scenario.                          |
| read_wal_buf_total_dur   | int8   | Total wait event duration of <b>wal_read_from_write_buffer</b> in the ultimate RTO scenario.                                |
| read_wal_buf_avg_dur     | int8   | Average wait event duration of <b>wal_read_from_write_buffer</b> in the ultimate RTO scenario.                              |
| read_wal_buf_min_dur     | int8   | Minimum wait event duration of <b>wal_read_from_write_buffer</b> in the ultimate RTO scenario.                              |
| read_wal_buf_max_dur     | int8   | Maximum wait event duration of <b>wal_read_from_write_buffer</b> in the ultimate RTO scenario.                              |

Example:

```
gaussdb=# SELECT * FROM gs_redo_stat_info();
-[RECORD 1]-----+-----
buffer_hit_rate | 70.5707
ddl_unlink_nrels_count | 3
read_buffer_io_counter | 1732
read_buffer_io_total_dur | 2850806
```

```
read_buffer_io_avg_dur | 1645
read_buffer_io_min_dur | 3
read_buffer_io_max_dur | 981639
read_wal_buf_counter | 9779
read_wal_buf_total_dur | 193612470
read_wal_buf_avg_dur | 19798
read_wal_buf_min_dur | 3
read_wal_buf_max_dur | 1914777
```

- gs\_recovery\_conflict\_waitevent\_info()

Description: Queries wait event information about the function that processes redo conflicts. The query must be executed by connecting to the standby DN.

Return type: SETOF record

| Name                          | Type | Description                                                   |
|-------------------------------|------|---------------------------------------------------------------|
| conflict_lock_counter         | int8 | Number of times that LOCK redo conflicts are triggered.       |
| conflict_lock_total_dur       | int8 | Total duration for processing LOCK redo conflicts.            |
| conflict_lock_avg_dur         | int8 | Average duration for processing LOCK redo conflicts.          |
| conflict_lock_min_dur         | int8 | Minimum duration for processing LOCK redo conflicts.          |
| conflict_lock_max_dur         | int8 | Maximum duration for processing LOCK redo conflicts.          |
| conflict_snapshot_counter     | int8 | Number of times that SNAPSHOT redo conflicts are triggered.   |
| conflict_snapshot_total_dur   | int8 | Total duration for processing SNAPSHOT redo conflicts.        |
| conflict_snapshot_avg_dur     | int8 | Average duration for processing SNAPSHOT redo conflicts.      |
| conflict_snapshot_min_dur     | int8 | Minimum duration for processing SNAPSHOT redo conflicts.      |
| conflict_snapshot_max_dur     | int8 | Maximum duration for processing SNAPSHOT redo conflicts.      |
| conflict_tablespace_counter   | int8 | Number of times that TABLESPACE redo conflicts are triggered. |
| conflict_tablespace_total_dur | int8 | Total duration for processing TABLESPACE redo conflicts.      |
| conflict_tablespace_avg_dur   | int8 | Average duration for processing TABLESPACE redo conflicts.    |
| conflict_tablespace_min_dur   | int8 | Minimum duration for processing TABLESPACE redo conflicts.    |

| Name                                     | Type | Description                                                              |
|------------------------------------------|------|--------------------------------------------------------------------------|
| conflict_tablespace_max_dur              | int8 | Maximum duration for processing TABLESPACE redo conflicts.               |
| conflict_database_counter                | int8 | Number of times that DATABASE redo conflicts are triggered.              |
| conflict_database_total_dur              | int8 | Total duration for processing DATABASE redo conflicts.                   |
| conflict_database_avg_dur                | int8 | Average duration for processing DATABASE redo conflicts.                 |
| conflict_database_min_dur                | int8 | Minimum duration for processing DATABASE redo conflicts.                 |
| conflict_database_max_dur                | int8 | Maximum duration for processing DATABASE redo conflicts.                 |
| conflict_truncate_counter                | int8 | Number of times that TRUNCATE redo conflicts are triggered.              |
| conflict_truncate_total_dur              | int8 | Total duration for processing TRUNCATE redo conflicts.                   |
| conflict_truncate_avg_dur                | int8 | Average duration for processing TRUNCATE redo conflicts.                 |
| conflict_truncate_min_dur                | int8 | Minimum duration for processing TRUNCATE redo conflicts.                 |
| conflict_truncate_max_dur                | int8 | Maximum duration for processing TRUNCATE redo conflicts.                 |
| conflict_standby_query_timeout_counter   | int8 | Number of times that STANDBY_QUERY_TIMEOUT redo conflicts are triggered. |
| conflict_standby_query_timeout_total_dur | int8 | Total duration for processing STANDBY_QUERY_TIMEOUT redo conflicts.      |
| conflict_standby_query_timeout_avg_dur   | int8 | Average duration for processing STANDBY_QUERY_TIMEOUT redo conflicts.    |
| conflict_standby_query_timeout_min_dur   | int8 | Minimum duration for processing STANDBY_QUERY_TIMEOUT redo conflicts.    |
| conflict_standby_query_timeout_max_dur   | int8 | Maximum duration for processing STANDBY_QUERY_TIMEOUT redo conflicts.    |

| Name                             | Type | Description                                                      |
|----------------------------------|------|------------------------------------------------------------------|
| conflict_force_recycle_counter   | int8 | Number of times that FORCE_RECYCLE redo conflicts are triggered. |
| conflict_force_recycle_total_dur | int8 | Total duration for processing FORCE_RECYCLE redo conflicts.      |
| conflict_force_recycle_avg_dur   | int8 | Average duration for processing FORCE_RECYCLE redo conflicts.    |
| conflict_force_recycle_min_dur   | int8 | Minimum duration for processing FORCE_RECYCLE redo conflicts.    |
| conflict_force_recycle_max_dur   | int8 | Maximum duration for processing FORCE_RECYCLE redo conflicts.    |

Example:

```
gaussdb=# SELECT * FROM gs_recovery_conflict_waitevent_info();
-[RECORD 1]-----+-----
conflict_lock_counter | 0
conflict_lock_total_dur | 0
conflict_lock_avg_dur | 0
conflict_lock_min_dur | 0
conflict_lock_max_dur | 0
conflict_snapshot_counter | 0
conflict_snapshot_total_dur | 0
conflict_snapshot_avg_dur | 0
conflict_snapshot_min_dur | 0
conflict_snapshot_max_dur | 0
conflict_tablespace_counter | 0
conflict_tablespace_total_dur | 0
conflict_tablespace_avg_dur | 0
conflict_tablespace_min_dur | 0
conflict_tablespace_max_dur | 0
conflict_database_counter | 0
conflict_database_total_dur | 0
conflict_database_avg_dur | 0
conflict_database_min_dur | 0
conflict_database_max_dur | 0
conflict_truncate_counter | 6
conflict_truncate_total_dur | 35872
conflict_truncate_avg_dur | 5978
conflict_truncate_min_dur | 5130
conflict_truncate_max_dur | 7459
conflict_standby_query_timeout_counter | 0
conflict_standby_query_timeout_total_dur | 0
conflict_standby_query_timeout_avg_dur | 0
conflict_standby_query_timeout_min_dur | 0
conflict_standby_query_timeoutmax_dur | 0
conflict_force_recycle_counter | 0
conflict_force_recycle_total_dur | 0
conflict_force_recycle_avg_dur | 0
conflict_force_recycle_min_dur | 0
conflict_force_recycle_max_dur | 0
```

- gs\_display\_delay\_ddl\_info()**

Description: Views information about files that are delayed for deletion on the standby node.

Return type: SETOF record

| Name       | Type | Description                                                                                                                |
|------------|------|----------------------------------------------------------------------------------------------------------------------------|
| type       | INT4 | Indicates that the deleted object is a table or database.                                                                  |
| lsn        | TEXT | Marks the location of a particular log file.                                                                               |
| tablespace | INT4 | Indicates the physical space for storing tables and indexes in a database.                                                 |
| database   | INT4 | Indicates the physical storage location of a database.                                                                     |
| relation   | INT4 | Indicates the object in a database, which can be the physical location of a table, view, or index.                         |
| bucketid   | INT4 | Specifies the bucket to which the relationship object belongs.                                                             |
| opt        | INT4 | Indicates the attribute of a compressed table.                                                                             |
| forknum    | INT4 | Specifies a suffix name for a subject name. You can find a unique physical file based on the subject name and suffix name. |

Example:

```
gaussdb=# SELECT * FROM gs_display_delay_ddl_info();
 type | lsn | tablespace | database | relation | bucketid | opt | forknum
-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)
```

## Functions for Collecting Statistics in Partitioned Tables

- gs\_stat\_get\_partition\_stats(oid)**  
 Description: Obtains the statistics of a specific partition.  
 Return type: record
- gs\_stat\_get\_xact\_partition\_stats(oid)**  
 Description: Obtains transaction statistics of a specific partition.  
 Return type: record
- gs\_stat\_get\_all\_partitions\_stats()**  
 Description: Obtains the statistics of all partitions.  
 Return type: SETOF record
- gs\_stat\_get\_xact\_all\_partitions\_stats()**  
 Description: Obtains transaction statistics of all partitions.  
 Return type: SETOF record
- gs\_statio\_get\_all\_partitions\_stats()**  
 Description: Obtains the I/O statistics of all partitions.

Return type: SETOF record

Examples of the preceding five functions

---

**⚠ CAUTION**

Statistics are reported asynchronously during execution. Based on UDP, delay and packet loss may occur during background thread processing. The following example is for reference only.

---

### Querying out-of-transaction statistics

```
gaussdb=# CREATE TABLE part_tab1
gaussdb=# (
gaussdb(# a int, b int
gaussdb(#)
gaussdb=# PARTITION BY RANGE(b)
gaussdb=# (
gaussdb(# PARTITION P1 VALUES LESS THAN(10),
gaussdb(# PARTITION P2 VALUES LESS THAN(20),
gaussdb(# PARTITION P3 VALUES LESS THAN(MAXVALUE)
gaussdb(#);
CREATE TABLE
gaussdb=# CREATE TABLE subpart_tab1
gaussdb=# (
gaussdb(# month_code VARCHAR2 (30) NOT NULL ,
gaussdb(# dept_code VARCHAR2 (30) NOT NULL ,
gaussdb(# user_no VARCHAR2 (30) NOT NULL ,
gaussdb(# sales_amt int
gaussdb(#)
gaussdb=# PARTITION BY RANGE (month_code) SUBPARTITION BY RANGE (dept_code)
gaussdb=# (
gaussdb(# PARTITION p_201901 VALUES LESS THAN('201903')
gaussdb(# (
gaussdb(# SUBPARTITION p_201901_a VALUES LESS THAN('2'),
gaussdb(# SUBPARTITION p_201901_b VALUES LESS THAN('3')
gaussdb(#),
gaussdb(# PARTITION p_201902 VALUES LESS THAN('201904')
gaussdb(# (
gaussdb(# SUBPARTITION p_201902_a VALUES LESS THAN('2'),
gaussdb(# SUBPARTITION p_201902_b VALUES LESS THAN('3')
gaussdb(#)
gaussdb(#);
CREATE TABLE
gaussdb=# CREATE INDEX index_part_tab1 ON part_tab1(b) LOCAL
gaussdb=# (
gaussdb(# PARTITION b_index1,
gaussdb(# PARTITION b_index2,
gaussdb(# PARTITION b_index3
gaussdb(#);
CREATE INDEX
gaussdb=# CREATE INDEX idx_user_no ON subpart_tab1(user_no) LOCAL;
CREATE INDEX
gaussdb=# INSERT INTO part_tab1 VALUES(1, 1);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 11);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 21);
INSERT 0 1
gaussdb=# UPDATE part_tab1 SET a = 2 WHERE b = 1;
UPDATE 1
gaussdb=# UPDATE part_tab1 SET a = 3 WHERE b = 11;
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(part_tab1) */ part_tab1 SET a = 4 WHERE b = 21;
UPDATE 1
gaussdb=# DELETE FROM part_tab1;
```

```

DELETE 3
gaussdb=# ANALYZE part_tab1;
ANALYZE
gaussdb=# VACUUM part_tab1;
VACUUM
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '1', '1', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '2', '2', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '1', '3', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '2', '4', 1);
INSERT 0 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 2 WHERE user_no='1';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 3 WHERE user_no='2';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 4 WHERE user_no='3';
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(subpart_tab1) */ subpart_tab1 SET sales_amt = 5 WHERE
user_no='4';
UPDATE 1
gaussdb=# DELETE FROM subpart_tab1;
DELETE 4
gaussdb=# ANALYZE subpart_tab1;
ANALYZE
gaussdb=# VACUUM subpart_tab1;
VACUUM
gaussdb=# SELECT * FROM gs_stat_all_partitions;
 partition_oid | schemaname | relname | partition_name | sub_partition_name | seq_scan |
seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del | n_tup_hot_upd | n_live_tup
|
 n_dead_tup | last_vacuum | last_autovacuum | last_analyze |
last_autoanalyze | vacuum_count | autovacuum_count | analyze_count | autoanalyze_count
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
16964 | public | subpart_tab1 | p_201902 | p_201902_b | 5 | 1 | 4
| 1 | 1 | 1 | 1 | 0 |
1 | 2023-05-15 20:36:45.293965+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688861+08 | 2000-01-01 08:00:00+08 | 1 | 0 | 1 | 0
16963 | public | subpart_tab1 | p_201902 | p_201902_a | 5 | 1 | 4
| 0 | 1 | 1 | 1 | 0 |
1 | 2023-05-15 20:36:45.291022+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688843+08 | 2000-01-01 08:00:00+08 | 1 | 0 | 1 | 0
16961 | public | subpart_tab1 | p_201901 | p_201901_b | 5 | 1 | 4
| 0 | 1 | 1 | 1 | 0 |
1 | 2023-05-15 20:36:45.288037+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688829+08 | 2000-01-01 08:00:00+08 | 1 | 0 | 1 | 0
16960 | public | subpart_tab1 | p_201901 | p_201901_a | 5 | 1 | 4
| 0 | 1 | 1 | 1 | 0 |
1 | 2023-05-15 20:36:45.285311+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688802+08 | 2000-01-01 08:00:00+08 | 1 | 0 | 1 | 0
16954 | public | part_tab1 | p3 | | 2 | 1 | 1
| 1 | 1 | 1 | 1 | 0 |
1 | 2023-05-15 20:36:29.490636+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:28.540115+08 | 2000-01-01 08:00:00+08 | 1 | 0 | 1 | 0
16953 | public | part_tab1 | p2 | | 4 | 1 | 1
| 1 | 1 | 1 | 1 | 0 |
1 | 2023-05-15 20:36:29.487914+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:28.540098+08 | 2000-01-01 08:00:00+08 | 1 | 0 | 1 | 0
16952 | public | part_tab1 | p1 | | 5 | 1 | 1
| 1 | 1 | 1 | 1 | 0 |
1 | 2023-05-15 20:36:29.48536+08 | 2000-01-01 08:00:00+08 | 2023-05-15 20:36:28.540071+08
| 2000-01-01 08:00:00+08 | 1 | 0 | 1 | 0
(7 rows)
gaussdb=# SELECT * FROM gs_statio_all_partitions;

```

| partition_oid | schemaname | relname      | partition_name | sub_partition_name | heap_blks_read | heap_blks_hit | idx_blks_read | idx_blks_hit | toast_blks_read | toast_blks_hit | tidx_blks_read | tidx_blks_hit |
|---------------|------------|--------------|----------------|--------------------|----------------|---------------|---------------|--------------|-----------------|----------------|----------------|---------------|
| 16964         | public     | subpart_tab1 | p_201902       | p_201902_b         | 4              | 8             |               |              |                 |                |                |               |
| 16963         | public     | subpart_tab1 | p_201902       | p_201902_a         | 4              | 8             |               |              |                 |                |                |               |
| 16961         | public     | subpart_tab1 | p_201901       | p_201901_b         | 4              | 8             |               |              |                 |                |                |               |
| 16960         | public     | subpart_tab1 | p_201901       | p_201901_a         | 4              | 8             |               |              |                 |                |                |               |
| 16954         | public     | part_tab1    | p3             |                    | 4              | 8             |               |              |                 |                |                | 2             |
| 16953         | public     | part_tab1    | p2             |                    | 4              | 8             |               |              |                 |                |                | 2             |
| 16952         | public     | part_tab1    | p1             |                    | 4              | 8             |               |              |                 |                |                | 2             |

(7 rows)

```
gaussdb=# SELECT * FROM gs_stat_get_partition_stats(16952);
partition_oid | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del |
n_tup_hot_upd | n_live_tup | n_dead_tup | last_vacuum | last_autovacuum |
last_analyze | last_autoanalyze | vacuum_count | autovacuum_count |
analyze_count | autoanalyze_count | last_data_changed | heap_blks_read | heap_blks_hit |
idx_blks_read | idx_blks_hit | tup_fetch | block_fetch
```

|       |                              |                        |                        |   |   |   |   |   |   |   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|-------|------------------------------|------------------------|------------------------|---|---|---|---|---|---|---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| 16952 | 5                            | 1                      | 1                      | 0 | 1 | 1 | 1 | 1 | 1 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1     | 2023-05-15 20:36:29.48536+08 | 2000-01-01 08:00:00+08 | 2000-01-01 08:00:00+08 | 1 | 0 | 1 |   |   |   |   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0     | 2000-01-01 08:00:00+08       | 4                      | 8                      |   |   |   |   |   |   |   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2     | 21                           | 0                      | 12                     |   |   |   |   |   |   |   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

(1 row)

### Querying statistics within a transaction

```
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO part_tab1 VALUES(1, 1);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 11);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 21);
INSERT 0 1
gaussdb=# UPDATE part_tab1 SET a = 2 WHERE b = 1;
UPDATE 1
gaussdb=# UPDATE part_tab1 SET a = 3 WHERE b = 11;
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(part_tab1) */ part_tab1 SET a = 4 WHERE b = 21;
UPDATE 1
gaussdb=# DELETE FROM part_tab1;
DELETE 3
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '1', '1', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '2', '2', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '1', '3', 1);
```

```

INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '2', '4', 1);
INSERT 0 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 2 WHERE user_no='1';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 3 WHERE user_no='2';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 4 WHERE user_no='3';
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(subpart_tab1) */ subpart_tab1 SET sales_amt = 5 WHERE
user_no='4';
UPDATE 1
gaussdb=# DELETE FROM subpart_tab1;
DELETE 4
gaussdb=# SELECT * FROM gs_stat_xact_all_partitions;
 partition_oid | schemaname | relname | partition_name | sub_partition_name | seq_scan |
seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del | n_tup_hot_upd
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
16964 | public | subpart_tab1 | p_201902 | p_201902_b | 4 | 4 | 1
| 2 | 1 | 1 | 1 | 1 |
16963 | public | subpart_tab1 | p_201902 | p_201902_a | 4 | 4 | 1
| 0 | 1 | 1 | 1 | 1 |
16961 | public | subpart_tab1 | p_201901 | p_201901_b | 4 | 4 | 1
| 0 | 1 | 1 | 1 | 1 |
16960 | public | subpart_tab1 | p_201901 | p_201901_a | 4 | 4 | 1
| 0 | 1 | 1 | 1 | 1 |
16954 | public | part_tab1 | p3 | | 1 | 1 | 1 | 2
| 1 | 1 | 1 | 1 |
16953 | public | part_tab1 | p2 | | 3 | 2 | 0 | 0
| 1 | 1 | 1 | 1 |
16952 | public | part_tab1 | p1 | | 4 | 2 | 0 | 0
| 1 | 1 | 1 | 1 |
(7 rows)

gaussdb=# SELECT * FROM gs_stat_get_xact_partition_stats(16952);
 partition_oid | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del |
n_tup_hot_upd | tup_fetch
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
16952 | 4 | 2 | 0 | 0 | 1 | 1 | 1 | 1 | 0
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE part_tab1;
DROP TABLE
gaussdb=# DROP TABLE subpart_tab1;
DROP TABLE

```

- gs\_stat\_get\_partition\_analyze\_count(oid)**  
Description: Specifies the number of times that a user starts analysis on a partition.  
Return type: bigint
- gs\_stat\_get\_partition\_autoanalyze\_count(oid)**  
Description: Specifies the number of times that the autovacuum daemon starts analysis in a partition.  
Return type: bigint
- gs\_stat\_get\_partition\_autovacuum\_count(oid)**  
Description: Specifies the number of times that the autovacuum daemon starts vacuum in a partition.  
Return type: bigint
- gs\_stat\_get\_partition\_last\_analyze\_time(oid)**

Description: Specifies the last time when a partition starts to be analyzed manually or by the autovacuum thread.

Return type: timestampz

- `gs_stat_get_partition_last_autoanalyze_time(oid)`

Description: Specifies the time when the last analysis initiated by the autovacuum daemon in a partition.

Return type: timestampz

- `gs_stat_get_partition_last_autovacuum_time(oid)`

Description: Specifies the time of the last vacuum initiated by the autovacuum daemon in a partition.

Return type: timestampz

- `gs_stat_get_partition_last_data_changed_time(oid)`

Description: Specifies the last time of a modification in a partition, such as insert, update, delete, and truncate. Currently, this parameter is not supported.

Return type: timestampz

- `gs_stat_get_partition_last_vacuum_time(oid)`

Description: Specifies the most recent time when the user manually cleared a table or when the autovacuum thread was started to clear a partition.

Return type: timestampz

- `gs_stat_get_partition_numscans(oid)`

Description: Specifies the number of rows scanned and read in partition order.

Return type: bigint

- `gs_stat_get_partition_tuples_returned(oid)`

Description: Specifies the number of sequential row scans in a partition.

Return type: bigint

- `gs_stat_get_partition_tuples_fetched(oid)`

Description: Specifies the number of rows fetched by bitmap scans in a partition.

Return type: bigint

- `gs_stat_get_partition_vacuum_count(oid)`

Description: Specifies the number of times that a user starts vacuum in a partition.

Return type: bigint

- `gs_stat_get_xact_partition_tuples_fetched(oid)`

Description: Specifies the number of tuple rows scanned in a transaction.

Return type: bigint

- `gs_stat_get_xact_partition_numscans(oid)`

Description: Specifies the number of sequential scans performed on a partition in the current transaction.

Return type: bigint

- `gs_stat_get_xact_partition_tuples_returned(oid)`

Description: Specifies the number of rows read through sequential scans in a partition in the current transaction.

Return type: bigint

- `gs_stat_get_partition_blocks_fetched(oid)`

Description: Specifies the number of disk block fetch requests for a partition.

Return type: bigint

- `gs_stat_get_partition_blocks_hit(oid)`

Description: Specifies the number of disk block requests found in cache for a partition.

Return type: bigint

- `pg_stat_get_partition_tuples_inserted(oid)`

Description: Specifies the number of rows in the corresponding table partition.

Return type: bigint

- `pg_stat_get_partition_tuples_updated(oid)`

Description: Specifies the number of rows that have been updated in the corresponding table partition.

Return type: bigint

- `pg_stat_get_partition_tuples_deleted(oid)`

Description: Specifies the number of rows deleted from the corresponding table partition.

Return type: bigint

- `pg_stat_get_partition_tuples_changed(oid)`

Description: Specifies the total number of inserted, updated, and deleted rows after a table partition was last analyzed or autoanalyzed.

Return type: bigint

- `pg_stat_get_partition_live_tuples(oid)`

Description: Specifies the number of live rows in a partitioned table.

Return type: bigint

- `pg_stat_get_partition_dead_tuples(oid)`

Description: Specifies the number of dead rows in a partitioned table.

Return type: bigint

- `pg_stat_get_xact_partition_tuples_inserted(oid)`

Description: Specifies the number of inserted tuples in the active sub-transactions related to a table partition.

Return type: bigint

- `pg_stat_get_xact_partition_tuples_deleted(oid)`

Description: Specifies the number of deleted tuples in the active sub-transactions related to a table partition.

Return type: bigint

- `pg_stat_get_xact_partition_tuples_hot_updated(oid)`

Description: Specifies the number of hot updated tuples in the active sub-transactions related to a table partition.

Return type: bigint

- `pg_stat_get_xact_partition_tuples_updated(oid)`  
Description: Specifies the number of updated tuples in the active sub-transactions related to a table partition.  
Return type: bigint
- `pg_stat_get_partition_tuples_hot_updated(oid)`  
Description: Returns statistics on the number of hot updated tuples in a partition with a specified partition ID.  
Parameter: oid  
Return type: bigint

## 7.5.27 Trigger Functions

- `pg_get_triggerdef(oid)`  
Description: Obtains the definition information of a trigger.  
Parameter: OID of the trigger to be queried  
Return type: text

```
-- Create the tri_insert table.
gaussdb=# CREATE TABLE tri_insert (a int, b int) distribute by hash(a);
CREATE TABLE
-- Create the trigger_func function.
gaussdb=# CREATE FUNCTION trigger_func() RETURNS trigger LANGUAGE plpgsql AS '
BEGIN
RAISE NOTICE 'trigger_func(%) called: action = %, when = %, level = %', TG_ARGV[0], TG_OP,
TG_WHEN, TG_LEVEL;
RETURN NULL;
END;';
CREATE FUNCTION
-- Create the before_ins_stmt_trig trigger.
gaussdb=# CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH
STATEMENT EXECUTE PROCEDURE trigger_func('before_ins_stmt');
CREATE TRIGGER
-- Create the after_ins_when_trig trigger.
gaussdb=# CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN
(new.a IS NOT NULL) EXECUTE PROCEDURE trigger_func('after_ins_when');
CREATE TRIGGER
-- View the trigger definition information of the tri_insert table.
gaussdb=# SELECT pg_get_triggerdef(oid) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;

pg_get_triggerdef

CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN ((new.a IS
NOT NULL)) EXECUTE PROCEDURE trigger_func('after_ins_when')
CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT
EXECUTE PROCEDURE trigger_func('before_ins_stmt')
(2 rows)
```

- `pg_get_triggerdef(oid, boolean)`  
Description: Obtains the definition information of a trigger.  
Parameter: OID of the trigger to be queried and whether it is displayed in pretty mode

### NOTE

Boolean parameters take effect only when the WHEN condition is specified during trigger creation.

Return type: text

Example:

```
-- View the trigger definition information of the tri_insert table in non-pretty mode.
gaussdb=# SELECT pg_get_triggerdef(oid, false) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;

pg_get_triggerdef

CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN ((new.a IS
NOT NULL)) EXECUTE PROCEDURE trigger_func('after_ins_when')
CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT
EXECUTE PROCEDURE trigger_func('before_ins_stmt')
(2 rows)

-- View the trigger definition information of the tri_insert table in pretty mode.
gaussdb=# SELECT pg_get_triggerdef(oid, true) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;

pg_get_triggerdef

CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN (new.a IS
NOT NULL) EXECUTE PROCEDURE trigger_func('after_ins_when')
CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT
EXECUTE PROCEDURE trigger_func('before_ins_stmt')
(2 rows)

-- Clear the tri_insert table.
gaussdb=# DROP TABLE tri_insert CASCADE;
DROP TABLE
-- Clear the trigger_func function.
gaussdb=# DROP FUNCTION trigger_func;
DROP FUNCTION
```

## 7.5.28 Hash Function

- bucketabstime (value, flag)

Description: Hashes the value in the abstime format and finds the corresponding hash bucket.

Parameter: **value** indicates the value to be converted, which is of the abstime type. **flag** is of the int type, indicating the data distribution mode. The value **0** indicates hash distribution.

Return type: int32

Example:

```
gaussdb=# select bucketabstime('2011-10-01 10:10:10.112',1);
bucketabstime

13954
(1 row)
```

- bucketbool (value, flag)

Description: Hashes the value in the bool format and finds the corresponding hash bucket.

Parameter: **value** indicates the value to be converted, which is of the bool type. **flag** is of the int type, indicating the data distribution mode. The value **0** indicates hash distribution.

Return type: int32

Example:

```
gaussdb=# select bucketbool(true,1);
bucketbool

1
(1 row)
gaussdb=# select bucketbool(false,1);
```

```
bucketbool
```

```

0
(1 row)
```

- bucketbpchar(value, flag)

Description: Hashes the value in the bpchar format and finds the corresponding hash bucket.

Parameter: **value** indicates the value to be converted, which is of the bpchar type. **flag** is of the int type, indicating the data distribution mode. The value **0** indicates hash distribution.

Return type: int32

Example:

```
gaussdb=# select bucketbpchar('test',1);
bucketbpchar

9761
(1 row)
```

- bucketbytea (value, flag)

Description: Hashes the value in the bytea format and finds the corresponding hash bucket.

Parameter: **value** indicates the value to be converted, which is of the bytea type. **flag** is of the int type, indicating the data distribution mode. The value **0** indicates hash distribution.

Return type: int32

Example:

```
gaussdb=# select bucketbytea('test',1);
bucketbytea

9761
(1 row)
```

- bucketcash (value, flag)

Description: Hashes the value in the money format and finds the corresponding hash bucket.

Parameter: **value** indicates the value to be converted, which is of the money type. **flag** is of the int type, indicating the data distribution mode. The value **0** indicates hash distribution.

Return type: int32

Example:

```
gaussdb=# select bucketcash(10::money,1);
bucketcash

8468
(1 row)
```

- getbucket (value, flag)

Description: Obtains the hash bucket from the distribution key.

**value** indicates the value to be entered, which can be of the following types: "char", abstime, bigint, boolean,bytea, character varying, character, date, double precision, int2vector, integer, interval, money, name, numeric, nvarchar2, oid, oidvector, raw, real, record, reltime, smalldatetime, smallint,text, time with time zone, time without time zone, timestamp with time zone, timestamp without time zone, tinyint, uuid

**flag** is of the int type, indicating the data distribution mode.

Return type: integer

Example:

```
gaussdb=# select getbucket(10,'H');
getbucket

 14535
(1 row)

gaussdb=# select getbucket(11,'H');
getbucket

 13449
(1 row)

gaussdb=# select getbucket(11,'R');
getbucket

 13449
(1 row)

gaussdb=# select getbucket(12,'R');
getbucket

 9412
(1 row)
```

- `ora_hash(expression,[seed])`

Description: Calculates the hash value of a given expression. **expression**: The value can be a character string, time, or number. The hash value is calculated based on the expression. **seed**: an int8 value that can return different results for the same input value. This parameter is optional and is used to calculate the hash value with a random number.

Return type: hash value of the int8 type.

Example:

```
gaussdb=# select ora_hash(123);
ora_hash

4089882933
(1 row)
gaussdb=# select ora_hash('123');
ora_hash

2034089965
(1 row)
gaussdb=# select ora_hash('sample');
ora_hash

1573005290
(1 row)
gaussdb=# select ora_hash(to_date('2012-1-2','yyyy-mm-dd'));
ora_hash

1171473495
(1 row)
gaussdb=# select ora_hash(123,234);
ora_hash

-9089505052966355682
(1 row)
gaussdb=# select ora_hash('123',234);
ora_hash

5742589019960764616
```

```
(1 row)
gaussdb=# select ora_hash('sample',234);
ora_hash

-1747984408055821656
(1 row)
gaussdb=# select ora_hash(to_date('2012-1-2','yyyy-mm-dd'),234);
ora_hash

-3306025179710572679
(1 row)
```

**NOTE**

This function is valid only when **a\_format\_version** is set to **10c** and **a\_format\_dev\_version** is set to **s2**.

- **hash\_array(anyarray)**

Description: Hashes an array, obtains the result of an array element using the hash function, and returns the combination result.

Parameter: data of the anyarray type.

Return type: integer

Example:

```
gaussdb=# select hash_array(ARRAY[[1,2,3],[1,2,3]]);
hash_array

-382888479
(1 row)
```

- **hash\_numeric(numeric)**

Description: Calculates the hash value of numeric data.

Parameter: data of the numeric type.

Return type: integer

Example:

```
gaussdb=# select hash_numeric(30);
hash_numeric

-282860963
(1 row)
```

- **hash\_range(anyrange)**

Description: Calculates the hash value of a range.

Parameter: data of the anyrange type.

Return type: integer

Example:

```
gaussdb=# select hash_range(numrange(1.1,2.2));
hash_range

683508754
(1 row)
```

- **hashbpchar(character)**

Description: Calculates the hash value of bpchar.

Parameter: data of the character type.

Return type: integer

Example:

```
gaussdb=# select hashbpchar('hello');
hashbpchar
```

```

-1870292951
(1 row)
```

- **hashchar(char)**

Description: Converts char and Boolean data into hash values.

Parameter: data of the char or bool type.

Return type: integer

Example:

```
gaussdb=# select hashbpchar('hello');
hashbpchar
```

```

-1870292951
(1 row)
```

```
gaussdb=# select hashchar('true');
hashchar
```

```

1686226652
(1 row)
```

- **hashenum(anyenum)**

Description: Converts enumerated values to hash values.

Parameter: data of the anyenum type.

Return type: integer

Example:

```
gaussdb=# CREATE TYPE b1 AS ENUM('good', 'bad', 'ugly');
CREATE TYPE
```

```
gaussdb=# call hashenum('good'::b1);
hashenum
```

```

1821213359
(1 row)
```

```
gaussdb=# DROP TYPE b1;
DROP TYPE
```

- **hashfloat4(real)**

Description: Converts float4 values to hash values.

Parameter: data of the real type.

Return type: integer

Example:

```
gaussdb=# select hashfloat4(12.1234);
hashfloat4
```

```

1398514061
(1 row)
```

- **hashfloat8(double precision)**

Description: Converts float8 values to hash values.

Parameter: data of the double precision type.

Return type: integer

Example:

```
gaussdb=# select hashfloat8(123456.1234);
hashfloat8
```

```

1673665593
(1 row)
```

- **hashinet(inet)**  
Description: Converts inet or cidr values to hash values.  
Parameter: data of the inet type.  
Return type: integer  
Example:

```
gaussdb=# select hashinet('127.0.0.1'::inet);
 hashinet

-1435793109
(1 row)
```
- **hashint1(tinyint)**  
Description: Converts INT1 values to hash values.  
Parameter: data of the tinyint type.  
Return type: uint32  
Example:

```
gaussdb=# select hashint1(20);
 hashint1

-2014641093
(1 row)
```
- **hashint2(smallint)**  
Description: Converts INT2 values to hash values.  
Parameter: data of the smallint type.  
Return type: uint32  
Example:

```
gaussdb=# select hashint2(20000);
 hashint2

-863179081
(1 row)
```
- **bucketchar**  
Description: Calculates the hash value of the input parameter.  
Parameter: char, integer  
Return type: integer
- **bucketdate**  
Description: Calculates the hash value of the input parameter.  
Parameter: date, integer  
Return type: integer
- **bucketfloat4**  
Description: Calculates the hash value of the input parameter.  
Parameter: real, integer  
Return type: integer
- **bucketfloat8**  
Description: Calculates the hash value of the input parameter.  
Parameters: double precision, integer  
Return type: integer

- bucketint1  
Description: Calculates the hash value of the input parameter.  
Parameter: tinyint, integer  
Return type: integer
- bucketint2  
Description: Calculates the hash value of the input parameter.  
Parameter: smallint, integer  
Return type: integer
- bucketint2vector  
Description: Calculates the hash value of the input parameter.  
Parameter: int2vector, integer  
Return type: integer
- bucketint4  
Description: Calculates the hash value of the input parameter.  
Parameter: integer, integer  
Return type: integer
- bucketint8  
Description: Calculates the hash value of the input parameter.  
Parameter: bigint, integer  
Return type: integer
- bucketinterval  
Description: Calculates the hash value of the input parameter.  
Parameter: interval, integer  
Return type: integer
- bucketname  
Description: Calculates the hash value of the input parameter.  
Parameter: name, integer  
Return type: integer
- bucketnumeric  
Description: Calculates the hash value of the input parameter.  
Parameter: numeric, integer  
Return type: integer
- bucketnvarchar2  
Description: Calculates the hash value of the input parameter.  
Parameter: nvarchar2, integer  
Return type: integer
- bucketoid  
Description: Calculates the hash value of the input parameter.  
Parameters: oid, integer  
Return type: integer

- bucketoidvector  
Description: Calculates the hash value of the input parameter.  
Parameter: oidvector, integer  
Return type: integer
- bucketraw  
Description: Calculates the hash value of the input parameter.  
Parameter: raw, integer  
Return type: integer
- bucketreltime  
Description: Calculates the hash value of the input parameter.  
Parameter: reltime, integer  
Return type: integer
- bucketsmalldatetime  
Description: Calculates the hash value of the input parameter.  
Parameter: smalldatetime, integer  
Return type: integer
- buckettext  
Description: Calculates the hash value of the input parameter.  
Parameter: text, integer  
Return type: integer
- buckettime  
Description: Calculates the hash value of the input parameter.  
Parameter: time without time zone, integer  
Return type: integer
- buckettimestamp  
Description: Calculates the hash value of the input parameter.  
Parameter: timestamp without time zone, integer  
Return type: integer
- buckettimestamptz  
Description: Calculates the hash value of the input parameter.  
Parameter: timestamp with time zone, integer  
Return type: integer
- buckettimetz  
Description: Calculates the hash value of the input parameter.  
Parameter: time with time zone, integer  
Return type: integer
- bucketuuid  
Description: Calculates the hash value of the input parameter.  
Parameter: uuid, integer  
Return type: integer

- bucketvarchar  
Description: Calculates the hash value of the input parameter.  
Parameter: character varying, integer  
Return type: integer

## 7.5.29 Prompt Message Function

- report\_application\_error()  
Description: This function can be used to throw errors during PL execution.  
Return type: void

**Table 7-126** Parameters for report\_application\_error

| Parameter | Type | Description                                                                           | Required |
|-----------|------|---------------------------------------------------------------------------------------|----------|
| log       | text | Content of an error message.                                                          | Yes      |
| code      | int4 | Error code corresponding to an error message. The value ranges from -20999 to -20000. | No       |

Example:

```
gaussdb=# CREATE OR REPLACE FUNCTION GET_RESULT_UNKNOWNN(
gaussdb(# IN context_id int, /*context_id*/
gaussdb(# IN col_type text /*col_type*/
gaussdb(#)RETURNS INTEGER
gaussdb=# AS $$
gaussdb$$ BEGIN
gaussdb$$ IF col_type IS NULL THEN
gaussdb$$ PG_CATALOG.REPORT_APPLICATION_ERROR('invalid input for the third parameter
col_type should not be null');
gaussdb$$ END IF;
gaussdb$$ PG_CATALOG.REPORT_APPLICATION_ERROR('UnSupport data type for
column_value(context: '||context_id||', '||PG_CATALOG.QUOTE_LITERAL(col_type)||')');
gaussdb$$ RETURN -1;
gaussdb$$ END;
gaussdb$$ $$ LANGUAGE plpgsql;
CREATE FUNCTION
gaussdb=# CALL GET_RESULT_UNKNOWNN(NULL, NULL);
ERROR: invalid input for the third parameter col_type should not be null
CONTEXT: SQL statement "CALL pg_catalog.report_application_error('invalid input for the third
parameter col_type should not be null')"
```

## 7.5.30 Fault Injection System Function

gs\_fault\_inject(int64, text, text, text, text, text)

Description: This function cannot be called. WARNING information "unsupported fault injection" is reported when this function is called, which does not affect or change the database.

Parameter: fault injection of the int64 type (0: CLOG extended page; 1: CLOG page reading; 2: forcible deadlock)

- If the first input parameter of text is set to **2** and the second input parameter of text is set to **1**, the second input parameter deadlock occurs. Other input parameters are not deadlocked. When the first input parameter is **0** or **1**, the second input parameter indicates the number of the start page from which the CLOG starts to be extended or read.
- The third input parameter of text indicates the number of extended or read pages when the first input parameter is **0** or **1**.
- The fourth to sixth input parameters of text are reserved.

Return type: int64

### 7.5.31 Redistribution Parameters

The following functions are system functions used by `gs_redis` during redistribution. Do not call them unless absolutely necessary.

- `pg_get_redis_rel_end_ctid(text, name, int, int)`
- `pg_get_redis_rel_start_ctid(text, name, int, int)`

Example:

```
gaussdb=# SELECT COUNT(1) FROM ONLY test
WHERE ctid BETWEEN pg_get_redis_rel_start_ctid(E'test':text,NULL::name,0,0) AND
pg_get_redis_rel_end_ctid(E'test':text,NULL::name,0,0);
count

0
(1 row)
```

- `pg_enable_redis_proc_cancelable()`

Example:

```
gaussdb=# select pg_enable_redis_proc_cancelable();
pg_enable_redis_proc_cancelable

t
```

- `pg_disable_redis_proc_cancelable()`

Example:

```
gaussdb=# select pg_disable_redis_proc_cancelable();
pg_disable_redis_proc_cancelable

t
```

- `pg_tupleid_get_blocknum(tid)`

Example:

```
gaussdb=# SELECT pg_tupleid_get_blocknum(ctid::tid) FROM test;
pg_tupleid_get_blocknum

0
(1 row)
```

- `pg_tupleid_get_offset(tid)`

Example:

```
gaussdb=# SELECT pg_tupleid_get_offset(ctid::tid) FROM test;
pg_tupleid_get_offset

1
(1 row)
```

- `pg_tupleid_get_ctid_to_bigint(ctid)`

Example:

```
gaussdb=# SELECT pg_tupleid_get_ctid_to_bigint(ctid::tid) FROM test;
pg_tupleid_get_ctid_to_bigint

1
(1 row)
```

## 7.5.32 Distribution Key Recommendation Functions

Distribution key recommendation is used to recommend distribution keys and distribution modes in a distributed database. The purpose is to reduce the labor cost of selecting distribution keys during service migration or rollout.

- `sqladvisor.init(char, boolean, boolean, boolean, int, int)`

Description: Initializes parameters.

Return type: Boolean

**Table 7-127** Parameter description of `init`

| Parameter                                | Type                 | Description                                                                                                                             | Mandatory or Not |
|------------------------------------------|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <code>kind</code>                        | <code>char</code>    | Recommendation type. Currently, this parameter can only be set to <b>D</b> .                                                            | Yes              |
| <code>isUseCost</code>                   | <code>boolean</code> | Indicates whether optimizers are used. If data is available, optimizers are used.                                                       | Yes              |
| <code>isUseCollect</code>                | <code>boolean</code> | Indicates whether the analysis is started from the collected load. The default value is <b>false</b> .                                  | No               |
| <code>isConstraint<br/>PrimaryKey</code> | <code>boolean</code> | Indicates whether primary key constraints are retained. The default value is <b>true</b> .                                              | No               |
| <code>sqlCount</code>                    | <code>int</code>     | Number of collected SQL statements. The default value is <b>10000</b> . The value ranges from 1 to 100000.                              | No               |
| <code>maxMemor<br/>y</code>              | <code>int</code>     | Maximum memory occupied by distribution key recommendation. The default value is <b>1024</b> . The value ranges from 1 to 10240, in MB. | No               |

- `sqladvisor.set_weight_params(real, real, real)`

Description: Sets the weight of different components in heuristic rules. A default parameter is set when the `init` function is invoked. This function does not need to be invoked during analysis.

Return type: Boolean

**Table 7-128** Parameter description of set\_weight\_params

| Parameter    | Type | Description                                           | Mandatory or Not |
|--------------|------|-------------------------------------------------------|------------------|
| joinWeight   | real | Weight of JOIN. The value ranges from 0 to 1000.      | Yes              |
| goupbyWeight | real | Weight of GROUP BY. The value ranges from 0 to 1000.  | Yes              |
| qualWeight   | real | Weight of predicate. The value ranges from 0 to 1000. | Yes              |

 **NOTE**

This function is optional. When the init function is executed, the default weights of JOIN, GROUP BY, and predicate are preset to 1.0, 0.1, and 0.05, respectively.

- sqladvisor.set\_cost\_params(bigint, boolean, text)

Description: Parameter that can be set in the Whtif cost model.

Return type: Boolean

**Table 7-129** Parameter description of set\_cost\_params

| Parameter     | Type    | Description                                                                                                                                                                                                                                                                       | Mandatory or Not |
|---------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| maxTime       | bigint  | Maximum recommendation duration, in minutes. If the value is less than or equal to <b>0</b> , the duration is not limited by default.                                                                                                                                             | Yes              |
| isTotalSQL    | boolean | Indicates whether all SQL statements are used for calculation. The value <b>true</b> indicates that all SQL statements are used for calculation. The value <b>false</b> indicates that SQL statements whose cost is too high or too low are filtered out based on the percentile. | Yes              |
| compressLevel | text    | Search space size of the recommendation algorithm. The options are <b>low</b> , <b>med</b> , and <b>high</b> .                                                                                                                                                                    | Yes              |

 **NOTE**

- This function is optional. When the init function is executed, **maxTime** is preset to **-1**, **isTotalSQL** is preset to **true**, and **compressLevel** is preset to **high**.
- A lower compression level indicates longer time, and it is more likely that a better result can be achieved.

- `sqladvisor.assign_table_type(text)`

Description: Specifies a table as a replication table.

Parameter: table name

Return type: Boolean

 **NOTE**

The specified replication table must be used before **analyze\_query** and **analyze\_workload** are invoked.

- `sqladvisor.analyze_query(text, int)`

Description: Imports SQL statements to be recommended and analyzes the components of the statements.

Return type: Boolean

**Table 7-130** Parameter description of `analyze_query`

| Parameter | Type | Description                                                                                                  | Mandatory or Not |
|-----------|------|--------------------------------------------------------------------------------------------------------------|------------------|
| query     | text | SQL statement                                                                                                | Yes              |
| frequency | int  | Frequency of a statement in the load. The default value is <b>1</b> . The value ranges from 1 to 2147483647. | No               |

 **NOTE**

- If the value of the **query** parameter contains special characters, such as single quotation marks ('), you can use single quotation marks (') to escape the special characters.
- This function is not supported in semi-online mode.
- `sqladvisor.analyze_workload()`

Description: Analyzes the load information collected online.

Return type: Boolean

- `sqladvisor.get_analyzed_result(text)`

Description: Obtains beneficial components extracted from the current table.

Parameter: text

Return type: record

The following table describes return fields.

| Name        | Type | Description |
|-------------|------|-------------|
| schema_name | text | Schema name |

| Name       | Type | Description                              |
|------------|------|------------------------------------------|
| table_name | text | Table name                               |
| col_name   | text | Column name                              |
| operator   | text | Operator type                            |
| count      | int  | Number of times that an operator is used |

- `sqladvisor.run()`

Description: Performs calculation and analysis based on the specified schema and input SQL statements.

Return type: Boolean

- `sqladvisor.get_distribution_key()`

Description: Obtains the recommendation result.

 **NOTE**

The analysis result is saved in a session. If the session disconnects, the result will be lost.

Return type: record

The following table describes return fields.

| Name              | Type      | Description                   |
|-------------------|-----------|-------------------------------|
| db_name           | text      | Database name                 |
| schema_name       | text      | Schema name                   |
| table_name        | text      | Table name                    |
| distribution_type | text      | Recommended distribution type |
| distribution_key  | text      | Recommended distribution key  |
| start_time        | timestamp | Recommended start time        |
| end_time          | timestamp | Recommended end time          |

| Name         | Type | Description                                        |
|--------------|------|----------------------------------------------------|
| cost_improve | text | Cost increase brought by the recommendation result |
| comment      | text | Comment                                            |

- sqladvisor.clean()

Description: Clears all the memory in the recommendation process of a session.

Return type: Boolean

- sqladvisor.start\_collect\_workload(int, int)

Description: Starts online load collection.

Return type: Boolean

**Table 7-131** Parameter description of start\_collect\_workload

| Parameter | Type | Description                                                                                                                            | Mandatory or Not |
|-----------|------|----------------------------------------------------------------------------------------------------------------------------------------|------------------|
| sqlCount  | int  | Maximum number of SQL statements for online load collection. The value ranges from 1 to 100000 and the default value is <b>10000</b> . | Yes              |
| maxMemory | int  | Maximum memory occupied by online load collection. The default value is <b>1024</b> . The value ranges from 1 to 10240, in MB.         | Yes              |

#### NOTICE

- The online collection function can be invoked only by the system administrator.
- The load of only one database can be collected at a time.
- Currently, only common SQL statements as well as DML and DQL statements in stored procedures are supported.

- sqladvisor.end\_collect\_workload()

Description: Disables online load collection.

Return type: Boolean

**NOTICE**

- The online collection function can be disabled only by the system administrator.

- `sqladvisor.clean_workload()`

Description: Clears the memory in the load.

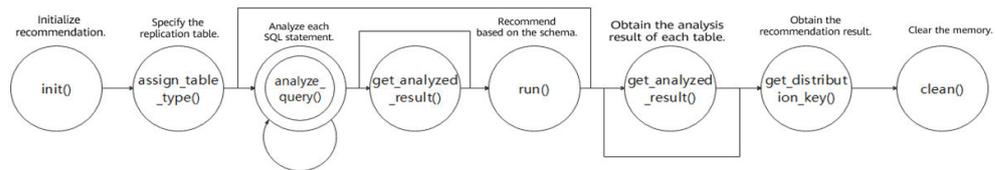
Return type: Boolean

**NOTICE**

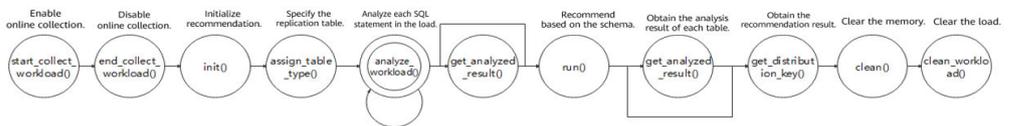
- The function of clearing memory in the load can be invoked only by the system administrator.
- You must manually execute the cleanup function.

**Suggestions**

- Invoke state machines in the heuristic or WhatIf cost recommendation mode.



- Invoke state machines in semi-online recommendation mode.



**7.5.33 Internal Functions**

The following functions of GaussDB use internal data types, which cannot be directly called by users.

- Selectivity calculation functions

|                          |                         |                          |                        |                        |                          |                  |
|--------------------------|-------------------------|--------------------------|------------------------|------------------------|--------------------------|------------------|
| areajoin<br>el           | areasel                 | arraycon<br>tjoin<br>sel | arraycon<br>tsel       | contjoin<br>el         | contsel                  | eqjoin<br>sel    |
| eqsel                    | iclikejoin<br>sel       | iclikesel                | icnlikejoin<br>sel     | icnlikese<br>l         | icregexe<br>qjoin<br>sel | icregexe<br>qsel |
| icregexn<br>ejoin<br>sel | icregexn<br>esel        | likejoin<br>el           | likesel                | neqjoin<br>el          | neqsel                   | nlikejoin<br>sel |
| nlikesel                 | positionj<br>oin<br>sel | position<br>el           | regexeqj<br>oin<br>sel | regexeq<br>el          | regexnej<br>oin<br>sel   | regexnes<br>el   |
| scalargtj<br>oin<br>sel  | scalargts<br>el         | scalartj<br>oin<br>sel   | scalartlts<br>el       | tsmatchj<br>oin<br>sel | tsmatchs<br>el           | -                |

- Statistics collection functions

|                               |                              |                                        |
|-------------------------------|------------------------------|----------------------------------------|
| array_tyanalyze               | range_tyanalyze              | ts_tyanalyze                           |
| local_rto_stat                | remote_rto_stat              | gs_plan_trace_delete                   |
| gs_plan_trace_watch_sqli<br>d | gs_plan_trace_show_sqli<br>s | standby_statement_<br>history_internal |

- Internal functions for sorting

|                        |                       |                      |                         |                           |
|------------------------|-----------------------|----------------------|-------------------------|---------------------------|
| bpchar_sorts<br>upport | bytea_sortsu<br>pport | date_sortsup<br>port | numeric_sort<br>support | timestamp_s<br>ortsupport |
|------------------------|-----------------------|----------------------|-------------------------|---------------------------|

- Internal type processing functions

|                    |                              |                            |                   |                              |                             |                                |
|--------------------|------------------------------|----------------------------|-------------------|------------------------------|-----------------------------|--------------------------------|
| abstimer<br>ecv    | euc_jis_2<br>004_to_<br>utf8 | int2recv                   | line_recv         | oidvecto<br>rrecv_ext<br>end | tidrecv                     | utf8_to_<br>koi8u              |
| anyarray<br>_recv  | euc_jp_t<br>o_mic            | int2vect<br>orrecv         | lseg_rec<br>v     | path_rec<br>v                | time_rec<br>v               | utf8_to_<br>shift_jis_<br>2004 |
| array_re<br>cv     | euc_jp_t<br>o_sjis           | int4recv                   | macaddr<br>_recv  | pg_node<br>_tree_rec<br>v    | time_tra<br>nsform          | utf8_to_<br>sjis               |
| ascii_to_<br>mic   | euc_jp_t<br>o_utf8           | int8recv                   | mic_to_a<br>scii  | point_re<br>cv               | timesta<br>mp_recv          | utf8_to_<br>uhc                |
| ascii_to_<br>utf8  | euc_kr_t<br>o_mic            | internal_<br>out           | mic_to_b<br>ig5   | poly_rec<br>v                | timesta<br>mp_tran<br>sform | utf8_to_<br>win                |
| big5_to_<br>euc_tw | euc_kr_t<br>o_utf8           | interval_<br>recv          | mic_to_e<br>uc_cn | pound_n<br>exttoken          | timesta<br>mptz_re<br>cv    | uuid_rec<br>v                  |
| big5_to_<br>mic    | euc_tw_t<br>o_big5           | interval_<br>transfor<br>m | mic_to_e<br>uc_jp | prsd_nex<br>ttoken           | timetz_r<br>ecv             | varbit_re<br>cv                |
| big5_to_<br>utf8   | euc_tw_t<br>o_mic            | iso_to_k<br>oi8r           | mic_to_e<br>uc_kr | range_re<br>cv               | tinterval<br>recv           | varbit_tr<br>ansform           |
| bit_recv           | euc_tw_t<br>o_utf8           | iso_to_m<br>ic             | mic_to_e<br>uc_tw | rawrecv                      | tsqueryr<br>ecv             | varchar_<br>transfor<br>m      |
| boolrecv           | float4rec<br>v               | iso_to_w<br>in1251         | mic_to_i<br>so    | record_r<br>ecv              | tsvectorr<br>ecv            | varcharr<br>ecv                |

|                                |                      |                   |                   |                                |                      |                   |
|--------------------------------|----------------------|-------------------|-------------------|--------------------------------|----------------------|-------------------|
| box_recv                       | float8recv           | iso_to_win866     | mic_to_koi8r      | regclassrecv                   | txid_snapshot_recv   | void_recv         |
| bpcharrecv                     | gb18030_to_utf8      | iso8859_1_to_utf8 | mic_to_latin1     | regconfigrecv                  | uhc_to_utf8          | win_to_utf8       |
| btoidsortsupport               | gbk_to_utf8          | iso8859_to_utf8   | mic_to_latin2     | regdictionaryrecv              | unknownrecv          | win1250_to_latin2 |
| bytearecv                      | -                    | johab_to_utf8     | mic_to_latin3     | regoperatorrecv                | utf8_to_ascii        | win1250_to_mic    |
| byteawithoutorderwithequalrecv | gtsvector_compress   | json_recv         | mic_to_latin4     | regoperrrecv                   | utf8_to_big5         | win1251_to_iso    |
| cash_recv                      | gtsvector_consistent | koi8r_to_iso      | mic_to_sjis       | regprocedurerecv               | utf8_to_euc_cn       | win1251_to_koi8r  |
| charrecv                       | gtsvector_decompress | koi8r_to_mic      | mic_to_win1250    | regprocrecv                    | utf8_to_euc_jis_2004 | win1251_to_mic    |
| cidr_recv                      | gtsvector_penalty    | koi8r_to_utf8     | mic_to_win1251    | regtyperrecv                   | utf8_to_euc_jp       | win1251_to_win866 |
| cidrecv                        | gtsvector_picksplit  | koi8r_to_win1251  | mic_to_win866     | reltimerrecv                   | utf8_to_euc_kr       | win866_to_iso     |
| circle_recv                    | gtsvector_same       | koi8r_to_win866   | namerecv          | shift_jis_2004_to_euc_jis_2004 | utf8_to_euc_tw       | win866_to_koi8r   |
| cstring_recv                   | gtsvector_union      | koi8u_to_utf8     | ngram_nexttoken   | shift_jis_2004_to_utf8         | utf8_to_gb18030      | win866_to_mic     |
| date_recv                      | hll_recv             | latin1_to_mic     | numeric_recv      | sjis_to_euc_jp                 | utf8_to_gbk          | win866_to_win1251 |
| domain_recv                    | hll_trans_recv       | latin2_to_mic     | numeric_transform | sjis_to_mic                    | utf8_to_iso8859      | xidrecv           |

|                                |                   |                      |                       |                        |                   |                          |
|--------------------------------|-------------------|----------------------|-----------------------|------------------------|-------------------|--------------------------|
| euc_cn_to_mic                  | -                 | latin2_to_win1250    | nvarchar2recv         | sjis_to_utf8           | utf8_to_iso8859_1 | xidrecv4                 |
| euc_cn_to_utf8                 | inet_recv         | latin3_to_mic        | oidrecv               | smalldatetime_recv     | utf8_to_johab     | xml_recv                 |
| euc_jis_2004_to_shift_jis_2004 | int1recv          | latin4_to_mic        | oidvectorrecv         | textrecv               | utf8_to_koi8r     | -                        |
| numeric_bool                   | int2vector_extend | int2vectorout_extend | int2vectorrecv_extend | int2vector_send_extend | int8_accum        | large_seq_rollback_ntree |
| large_seq_upgrade_ntree        | int16eq           | int16ge              | int16gt               | int16in                | int16le           | int16lt                  |
| int16mi                        | int16mul          | int16ne              | int16out              | int16pl                | int16recv         | int16send                |
| int16_bool                     | i16toi1           | anyset_in            | anyset_out            | btint2setcmp           | btint4setcmp      | btint8setcmp             |
| btsetcmp                       | btsetint2cmp      | btsetint4cmp         | btsetint8cmp          | btsetsortsupport       | float4            | float8                   |
| hashsetint                     | hashsetext        | int2                 | int2seteq             | int2setge              | int2setgt         | int2setle                |
| int2setlt                      | int2setne         | int4                 | int4seteq             | int4setge              | int4setgt         | int4setle                |
| int4setlt                      | int4setne         | int8                 | int8seteq             | int8setge              | int8setgt         | int8setle                |
| int8setlt                      | int8setne         | set                  | set_in                | set_out                | set_recv          | set_send                 |
| seteq                          | setge             | setgt                | setint2eq             | setint2ge              | setint2gt         | setint2le                |
| setint2lt                      | setint2ne         | setint4eq            | setint4ge             | setint4gt              | setint4le         | setint4lt                |
| setint4ne                      | setint8eq         | setint8ge            | setint8gt             | setint8le              | setint8lt         | setint8ne                |
| setle                          | setlt             | setne                | settexteq             | settextge              | settextgt         | settextle                |
| settextlt                      | settextne         | settovarchar         | settonumber           | settonvarchar2         | settotext         | settovarchar             |

|                      |           |           |           |           |           |                      |
|----------------------|-----------|-----------|-----------|-----------|-----------|----------------------|
| textseteq            | textsetge | textsetgt | textsetle | textsetlt | textsetne | gb18030_2022_to_utf8 |
| utf8_to_gb18030_2022 | -         | -         | -         | -         | -         | -                    |

- Internal functions for aggregation operations

|                                   |                                |                                  |                              |                              |                                   |                                |
|-----------------------------------|--------------------------------|----------------------------------|------------------------------|------------------------------|-----------------------------------|--------------------------------|
| array_agg_finalfn                 | array_agg_transfn              | bytea_storing_agg_finalfn        | bytea_storing_agg_transfn    | date_list_agg_noarg2_transfn | date_list_agg_transfn             | float4_list_agg_noarg2_transfn |
| float4_list_agg_transfn           | float8_list_agg_noarg2_transfn | float8_list_agg_transfn          | int2_list_agg_noarg2_transfn | int2_list_agg_transfn        | int4_list_agg_noarg2_transfn      | int4_list_agg_transfn          |
| int8_list_agg_noarg2_transfn      | int8_list_agg_transfn          | interval_list_agg_noarg2_transfn | interval_list_agg_transfn    | list_agg_finalfn             | list_agg_noarg2_transfn           | list_agg_transfn               |
| median                            | median_float8_finalfn          | median_interval_finalfn          | median_transfn               | mode_final                   | numeric_list_agg_noarg2_transfn   | numeric_list_agg_transfn       |
| ordered_set_transition            | percentile_cont_float8_final   | percentile_cont_interval_final   | string_agg_finalfn           | string_agg_transfn           | timestamp_list_agg_noarg2_transfn | timestamp_list_agg_transfn     |
| timestamp_list_agg_noarg2_transfn | timestamp_list_agg_transfn     | checksumtext_agg_transfn         | json_agg_transfn             | json_agg_finalfn             | json_object_agg_transfn           | json_object_agg_finalfn        |

- Hash internal functions

|               |            |                |                |                  |              |                   |
|---------------|------------|----------------|----------------|------------------|--------------|-------------------|
| hashbeginscan | hashbuild  | hashbuildempty | hashbulkdelete | hashcostestimate | hashendscan  | hashgetbitmap     |
| hashgettuple  | hashinsert | hashmarkpos    | hashmerge      | hashrescan       | hashrestrpos | hashvacuumcleanup |
| hashvarlena   | jsonb_hash | -              | -              | -                | -            | -                 |

- Internal functions of the B-tree index

|              |                  |                    |                   |                   |                     |                     |
|--------------|------------------|--------------------|-------------------|-------------------|---------------------|---------------------|
| cbtreebuild  | cbtreecanreturn  | cbtreecostestimate | cbtreegetbitmap   | cbtreegettuple    | btbeginscan         | btbuild             |
| btbuildempty | btbulkdelete     | btcanreturn        | btcostestimate    | btendscan         | btfloat4sortsupport | btfloat8sortsupport |
| btgetbitmap  | btgettuple       | btinsert           | btint2sortsupport | btint4sortsupport | btint8sortsupport   | btmarkpos           |
| btmerge      | btnameortsupport | btrescan           | btrestropos       | bttextsortsupport | btvacuumcleanup     | cbtreeoptions       |

- Internal functions of the Psort index

|            |                |                   |                |               |
|------------|----------------|-------------------|----------------|---------------|
| psortbuild | psortcanreturn | psortcostestimate | psortgetbitmap | psortgettuple |
|------------|----------------|-------------------|----------------|---------------|

- Internal functions of the UB-tree index

|                  |            |               |               |              |
|------------------|------------|---------------|---------------|--------------|
| ubtbeginscan     | ubtbuild   | ubtbuildempty | ubtbulkdelete | ubtcanreturn |
| ubtcostestimate  | ubtendscan | ubtgetbitmap  | ubtgettuple   | ubtinsert    |
| ubtmarkpos       | ubtmerge   | ubtoptions    | ubtrescan     | ubtrestropos |
| ubtvacuumcleanup | -          | -             | -             | -            |

- plpgsql internal function

plpgsql\_inline\_handler

- Foreign table-related internal functions

|                  |                  |                       |                      |                  |                    |                 |
|------------------|------------------|-----------------------|----------------------|------------------|--------------------|-----------------|
| dist_fdw_handler | roach_handler    | streaming_fdw_handler | dist_fdw_validator   | file_fdw_handler | file_fdw_validator | log_fdw_handler |
| gc_fdw_handler   | gc_fdw_validator | dblink_fdw_handler    | dblink_fdw_validator | -                | -                  | -               |

- Internal function related to data skew optimization

distributed\_count

- Internal functions related to table statistics

|                                |                            |                                |                                 |
|--------------------------------|----------------------------|--------------------------------|---------------------------------|
| pgxc_get_stat_dir<br>ty_tables | pgxc_stat_dir<br>ty_tables | get_global_stat_a<br>ll_tables | get_summary_stat_<br>all_tables |
|--------------------------------|----------------------------|--------------------------------|---------------------------------|

- Functions for reading data remotely  

`gs_read_block_from_remote` is used to read the pages of a table file. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.
- Function for reading files remotely  

`gs_read_file_size_from_remote` is used to read the size of a specified file. Before using the `gs_repair_file` function to repair a file, you need to obtain the size of the file from the remote end to verify the missing file information and repair the missing files one by one. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

`gs_read_file_from_remote` is used to read a specified file. After obtaining the file size by using the `gs_read_file_size_from_remote` function, `gs_repair_file` reads the remote file segment by segment using this function. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.
- Auxiliary functions for incrementally rebuilding other standby or cascaded standby DN's using the standby DN.  

`gs_standby_incremental_filemap_create` is used to create a temporary filemap file for incremental rebuilding on the standby DN. The filemap file is used to store the path and size of the data to be transferred during the incremental rebuilding. This API can be called only when the initial user is used and **application** is `gs_rewind`.

`gs_standby_incremental_filemap_insert` is used to insert file information into a specified temporary filemap file. The file information includes the file path, transfer start point, length of data to be transferred at a time, and rebuild flag bit. This API can be called only when the initial user is used and **application** is `gs_rewind`.

`gs_standby_incremental_filemap_execute` is used to obtain file information stored in a specified temporary filemap file and delete the specified filemap file for data transmission during incremental standby DN rebuilding. This API can be called only when the initial user is used and **application** is `gs_rewind`.
- View-related reference functions  

`adm_hist_sqlstat_func`  
`adm_hist_sqlstat_idlog_func`  
`adm_hist_sqltext_func`
- `gs_txn_snapshot` system catalog maintenance function  

`gs_insert_delete_txn_snapshot` is used to maintain the `gs_txn_snapshot` system catalog of each node in distributed GTM-Lite mode. Only system administrators can call `gs_insert_delete_txn_snapshot`.
- XMLType functions  

`isschemavalid` (unavailable)

## 7.5.34 AI Feature Functions

- `gs_index_advise(text)`  
Description: Recommends an index for a single query statement.  
Parameter: SQL statement string  
Return type: record  
For details, see section "Index-advisor: Index Recommendation > Single-Query Index Recommendation" in *Feature Guide*.
- `hypopg_create_index(text, [text])`  
Description: Creates a virtual index.  
Parameter: character string of the statement for creating an index, level of the created virtual index (optional)  
Return type: record  
For details, see section "Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.
- `hypopg_display_index([text])`  
Description: Displays information about all created virtual indexes.  
Parameter: level of the virtual index to be displayed (optional)  
Return type: record  
For details, see section "Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.
- `hypopg_drop_index(oid)`  
Description: Deletes a specified virtual index.  
Parameter: OID of the index  
Return type: Boolean  
For details, see section "Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.
- `hypopg_reset_index([text])`  
Description: Clears all virtual indexes.  
Parameter: level of the virtual index to be cleared (optional)  
Return type: none  
For details, see section "Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.
- `hypopg_estimate_size(oid)`  
Description: Estimates the space required for creating a specified index.  
Parameter: OID of the index  
Return type: int8  
For details, see section "Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.
- `db4ai_predict_by_bool (text, VARIADIC "any")`  
Description: Obtains a model whose return value is of the Boolean type for model inference. This function is an internal function. You are advised to use the **PREDICT BY** syntax for inference.

Parameter: model name and input column name of the inference task

Return type: Boolean

 **NOTE**

This function is an internal function. You are advised not to use it directly.

- `db4ai_predict_by_float4(text, VARIADIC "any")`

Description: Obtains a model whose return value is of the float4 type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

Parameter: model name and input column name of the inference task

Return type: float

 **NOTE**

This function is an internal function. You are advised not to use it directly.

- `db4ai_predict_by_float8(text, VARIADIC "any")`

Description: Obtains a model whose return value is of the float8 type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

Parameter: model name and input column name of the inference task

Return type: float

 **NOTE**

This function is an internal function. You are advised not to use it directly.

- `db4ai_predict_by_int32(text, VARIADIC "any")`

Description: Obtains a model whose return value is of the int32 type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

Parameter: model name and input column name of the inference task

Return type: int

 **NOTE**

This function is an internal function. You are advised not to use it directly.

- `db4ai_predict_by_int64(text, VARIADIC "any")`

Description: Obtains a model whose return value is of the int64 type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

Parameter: model name and input column name of the inference task

Return type: int

 **NOTE**

This function is an internal function. You are advised not to use it directly.

- `db4ai_predict_by_numeric(text, VARIADIC "any")`

Description: Obtains a model whose return value is of the numeric type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

Parameter: model name and input column name of the inference task

Return type: numeric

 **NOTE**

This function is an internal function. You are advised not to use it directly.

- `db4ai_predict_by_text(text, VARIADIC "any")`

Description: Obtains a model whose return value is of the character type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

Parameter: model name and input column name of the inference task

Return type: text

 **NOTE**

This function is an internal function. You are advised not to use it directly.

- `db4ai_predict_by_float8_array(text, VARIADIC "any")`

Description: Obtains a model whose return value is of the character type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

Parameter: model name and input column name of the inference task

Return type: text

 **NOTE**

This function is an internal function. You are advised not to use it directly.

- `gs_explain_model(text)`

Description: Obtains the model whose return value is of the character type for text-based model parsing.

Parameter: model name

Return type: text

 **NOTE**

This function is an internal function and is unavailable in the current version.

- `gs_ai_stats_explain(text, text[])`

Description: Prints the intelligent statistics of multiple columns in the corresponding table and columns on the current node.

Parameters: table name and column name collection.

Return type: text

Example:

```
gaussdb=# SET enable_ai_stats=1;
gaussdb=# drop table db4ai_bayesnet;
gaussdb=# create table db4ai_bayesnet(attr1 VARCHAR(256), attr2 VARCHAR(256));
gaussdb=# INSERT INTO db4ai_bayesnet SELECT 'x','x' FROM generate_series(1, 10000) AS s(i);
gaussdb=# ANALYZE db4ai_bayesnet((attr1,attr2));
gaussdb=# select gs_ai_stats_explain('db4ai_bayesnet', ARRAY['attr1', 'attr2']);
```

- `ai_watchdog_detection_warnings()`

Description: Obtains the risk alarm information of the AI watchdog.

Parameter: none

Return type: record

Example:

- ```
gaussdb=# select * from ai_watchdog_detection_warnings();
```
- `ai_watchdog_monitor_status(int)`
Description: Obtains the monitoring information of the AI watchdog.
Parameter: the upper limit of the length of the returned monitoring sequence. The value range is (0,100). The default value is **10**.
Return type: record
Example:

```
gaussdb=# select * from ai_watchdog_monitor_status();
```
 - `ai_watchdog_parameters()`
Description: Obtains the internal parameters or status information of the AI watchdog.
Parameter: none
Return type: record
Example:

```
gaussdb=# select * from ai_watchdog_parameters();
```

7.5.35 Dynamic Data Masking Functions

NOTE

This function is an internal function.

- `creditcardmasking(col text, letter char default 'x')`
Description: Replaces the digits before the last four bits following the col string with letters.
Parameters: string to be replaced and string used for replacement.
Return type: text
Example:

```
gaussdb=# select * from creditcardmasking('4511-8454-2178-6551', 'x');
creditcardmasking
-----
xxxx-xxxx-xxxx-6551
(1 row)
```
- `basicemailmasking(col text, letter char default 'x')`
Description: Replaces the characters before the first at sign (@) in the col string with letters.
Parameters: string to be replaced and string used for replacement.
Return type: text
Example:

```
gaussdb=# select * from basicemailmasking('Alex15@huawei.com', 'x');
basicemailmasking
-----
xxxxxx@huawei.com
(1 row)
```
- `fullemailmasking(col text, letter char default 'x')`
Description: Replaces the characters (except @) before the last period (.) in the col string with letters.
Parameters: string to be replaced and string used for replacement.
Return type: text

Example:

```
gaussdb=# select * from fullemailmasking('Alex15@huawei.com','x');
fullemailmasking
-----
xxxxxx@xxxxxx.com
(1 row)
```

- alldigitsmasking(col text, letter char default '0')

Description: Replaces the digits in the col string with letters.

Parameters: string to be replaced and string used for replacement.

Return type: text

Example:

```
gaussdb=# select * from alldigitsmasking('abcdef 123456 ui 323 jsfd321 j3k2l3','0');
alldigitsmasking
-----
abcdef 000000 ui 000 jsfd000 j0k0l0
(1 row)
```

- shufflemasking(col text)

Description: Sorts the characters in the col string out of order.

Parameters: string to be replaced and string used for replacement.

Return type: text

Example:

```
gaussdb=# select * from shufflemasking('abcdef 123456 ui 323 jsfd321 j3k2l3');
shufflemasking
-----
22dc3316 3jb af4e3f135sjl ud2 k32i
(1 row)
```

- randommasking(col text)

Description: Randomizes the characters in the col string.

Parameters: string to be replaced and string used for replacement.

Return type: text

Example:

```
gaussdb=# select * from randommasking('abcdef');
randommasking
-----
63d8dc
(1 row)
```

- regexprmasking(col text, reg text, replace_text text, pos INTEGER default 0, reg_len INTEGER default -1)

Description: Replaces the col string with a regular expression.

Parameters: string to be replaced, regular expression, string used for replacement, start position for replacement, and replacement length.

Return type: text

Example:

```
gaussdb=# select * from regexprmasking('abcdef 123456 ui 323 jsfd321 j3k2l3','[\d+]','0');
regexprmasking
-----
abcdef 000000 ui 000 jsfd000 j0k0l0
(1 row)
```


parameter is set to **true**, it is a reserved parameter and not supported currently. By default, only the initial user, users with the SYSADMIN attribute, and users with the OPRADMIN attribute in the O&M mode can view the information. Other users can view the information only after being granted permissions.

The returned result is as follows:

- **rel_oid** and **rel_name** indicate the table OID and table name of the corresponding file, and **miss_file_path** indicates the relative path of the lost file.

Parameters:

- **verify_segment**
Specifies the range of files to be checked.
The value can be **true** or **false** (default value). **true** is a reserved parameter and is not supported currently.

Return type: record

Example (The abnormal line is displayed only when an exception is detected. Otherwise, no line is displayed.):

```
gaussdb=# select * from gs_verify_data_file();
node_name      | rel_oid | rel_name  | miss_file_path
-----+-----+-----+-----
dn_6001_6002_6003 | 16554 | test    | base/16552/24745
```

- **gs_repair_file(tableoid Oid, path text, timeout int)**

Description: Repairs the file based on the input parameters. Only the primary DN with normal primary/standby connection is supported. The parameter is set based on the OID and path returned by the `gs_verify_data_file` function. If the repair is successful, **true** is returned. If the repair fails, the failure cause is displayed. By default, only the initial user, users with the SYSADMIN attribute, and users with the OPRADMIN attribute in the O&M mode on the primary DN can view the information. Other users can view the information only after being granted permissions.

CAUTION

1. If a file on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal.
2. If a file exists but its size is 0, the file will not be repaired. To repair the file, you need to delete the file whose size is 0 and then repair it.
3. You can delete a file only after the file descriptor is automatically closed. You can manually restart the process or perform a primary/standby switchover.

Parameters:

- **tableoid**
OID of the table corresponding to the file to be repaired. Set this parameter based on the **rel_oid** column in the list returned by the `gs_verify_data_file` function.

Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

- path

Path of the file to be repaired. Set this parameter based on the **miss_file_path** column in the list returned by the `gs_verify_data_file` function.

Value range: a string

- timeout

Specifies the duration for waiting for the standby DN to replay. The repair file needs to wait for the standby DN to be put back to the corresponding location on the current primary DN. Set this parameter based on the replay duration of the standby DN.

Value range: 60s to 3600s.

Return type: Boolean

Example (Set **tablespace** and **path** based on the output of `gs_verify_data_file`):

```
gaussdb=# select * from gs_repair_file(16554,'base/16552/24745',360);
gs_repair_file
-----
t
```

- `local_bad_block_info()`

Description: Displays the page damage of the instance. You can read the page from the disk and record the page CRC failure. By default, only the initial user, users with the SYSADMIN attribute, users with the MONADMIN attribute, users with the OPRADMIN attribute in the O&M mode, and monitor users can view the information. Other users can view the information only after being granted permissions. **file_path** indicates the relative path of the damaged file. **block_num** indicates the number of the page where the file is damaged. The page number starts from 0. **check_time** indicates the time when the page damage is detected. **repair_time** indicates the time when the page is repaired.

Return type: record

Example (Related entries are displayed only when there are damaged records. Otherwise, no log is displayed.):

```
gaussdb=# select * from local_bad_block_info();
node_name | spc_node | db_node | rel_node | bucket_node | fork_num | block_num | file_path |
check_time | repair_time
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
dn_6001_6002_6003 | 1663 | 16552 | 24745 | -1 | 0 | 0 | base/16552/24745 |
2022-01-13 20:19:08.385004+08 | 2022-01-13 20:19:08.407314+08
```

- `remote_bad_block_info()`

Description: Queries the page damage of other instances except the current instance when a query is performed on the CN. The recorded data is the same as that of the `local_bad_block_info` function executed on other instances. The execution result on the DN is empty. By default, only the initial user, users with the SYSADMIN attribute, users with the MONADMIN attribute, users with the OPRADMIN attribute in the O&M mode, and monitor users can view the information. Other users can view the information only after being granted permissions.

Return type: record

- `local_clear_bad_block_info()`

Description: Deletes data of repaired pages from `local_bad_block_info`, that is, information whose **repair_time** is not empty. By default, only the initial user, users with the SYSADMIN permission, users with the OPRADMIN attribute in the O&M mode, and monitor users can view the information. Other users can view the information only after being granted permissions.

Return type: bool

Example:

```
gaussdb=# select * from local_clear_bad_block_info();
result
-----
t
```

- `remote_clear_bad_block_info()`

Description: Clears the data of the repaired pages of other instances except the current instance when this function is executed on the CN, that is, information whose **repair_time** is not empty. The execution result on the DN is empty. By default, only the initial user, users with the SYSADMIN permission, users with the OPRADMIN attribute in the O&M mode, and monitor users can view the information. Other users can view the information only after being granted permissions.

Return type: record

- `gs_verify_and_tryrepair_page` (path text, blocknum Oid, verify_mem bool, is_segment bool)

Description: Verifies the page specified by the instance. By default, only the initial user, users with the SYSADMIN permission, and users with the OPRADMIN permission in the O&M mode on the primary DN can view the information. Other users can view the information only after being granted permissions. In the command output, **disk_page_res** indicates the verification result of the page on the disk, **mem_page_res** indicates the verification result of the page in the memory, and **is_repair** specifies whether the repair function is triggered during the verification. **t** indicates that the page is repaired, and **f** indicates that the page is not repaired.

Note:

- a. If a page on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal. Damaged pages of hash bucket tables cannot be repaired.
- b. The repair triggered by this function can only repair pages in the memory. The repair takes effect only after the memory pages are flushed to disk.

Parameters:

- path

Path of the damaged file. Set this parameter based on the **file_path** column in `local_bad_block_info`. To verify the undo pages of the Ustore table, enter the path of the undo pages to be verified.

Value range: a string

- blocknum

Page number of the damaged file. Set this parameter based on the **block_num** column in `local_bad_block_info`. To verify the undo pages of the Ustore table, enter the path of the undo pages to be verified.

Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

– `verify_mem`

Specifies whether to verify a specified page in the memory. If this parameter is set to **false**, only pages on the disk are verified. If this parameter is set to **true**, pages in the memory and on the disk are verified. If a page on the disk is damaged, the system verifies the basic information of the page in the memory and flushes the page to the disk to restore the page. If a page is not found in the memory during memory page verification, the page on the disk is read through the memory API. During this process, if the disk page is faulty, the remote read automatic repair function is triggered.

Value range: The value is of a Boolean type and can be **true** or **false**.

– `is_segment`

Specifies whether the table is a segment-page table. **false** indicates that the table is not a segment-page table. **true** indicates a reserved parameter value, which is not supported currently.

Value range: The value is of a Boolean type and can be **true** or **false**.

Return type: record

Examples (Transfer parameters based on the output of `local_bad_block_info`. Otherwise, an error is reported.):

```
gaussdb=# select * from gs_verify_and_tryrepair_page('base/16552/24745',0,false,false);
node_name | path | blocknum | disk_page_res | mem_page_res | is_repair
-----+-----+-----+-----+-----+-----
dn_6001_6002_6003 | base/16552/24745 | 0 | page verification succeeded. | | f
```

• `gs_repair_page(path text, blocknum oid is_segment bool, timeout int)`

Description: Repairs the specified page of the instance. This function can be used only on the primary DN that is properly connected to standby DNs. By default, only the initial user, users with the SYSADMIN permission, and users with the OPRADMIN permission in the O&M mode on the primary DN can view the information. Other users can view the information only after being granted permissions.

Note: If a page on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal. Damaged pages of hash bucket tables cannot be repaired.

Parameter description:

– `path`

Path of the damaged page. Set this parameter based on the **file_path** column in `local_bad_block_info` or the **path** column in the `gs_verify_and_tryrepair_page` function.

Value range: a string

– `blocknum`

Number of the damaged page. Set this parameter based on the **block_num** column in `local_bad_block_info` or the **blocknum** column in the `gs_verify_and_tryrepair_page` function.

Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

- `is_segment`

Specifies whether the table is a segment-page table. **false** indicates that the table is not a segment-page table. **true** indicates a reserved parameter value, which is not supported currently.

Value range: The value is of a Boolean type and can be **true** or **false**.

- `timeout`

Duration of waiting for standby DN replay. The page to be repaired needs to wait for the standby DN to be played back to the location of the current primary DN. Set this parameter based on the replay duration of the standby DN.

Value range: 60s to 3600s.

Return type: Boolean

Examples (Transfer parameters based on the output of `local_bad_block_info`. Otherwise, an error is reported.):

```
gaussdb=# select * from gs_repair_page('base/16552/24745',0,false,60);
result
-----
t
```

- `gs_edit_page_bypath`(`path` text, `blocknum` int64, `offset` int, `data` text, `data_size` int, `read_backup` bool, `storage_type` text)

Description: Specifies the path, block number, offset, target data to be modified, and length of the target table file, and transfers the target data to the corresponding column on the page. The **read_backup** column determines the file read mode, and the **storage_type** column indicates the file storage mode (for example, page storage). To prevent modification by mistake, this function does not directly modify the original page but modifies the copied page and flushes the modified page to the specified path. Only the SYSADMIN or OPRADMIN in O&M mode can execute this function. In a distributed environment, this function must be executed on DNs.

Return type: text

Table 7-132 gs_edit_page_bypath parameters

Category	Parameter	Type	Description
Input parameter	path	text	<p>Physical file path of the file to be modified, which is related to the read_backup field. The value can be the relative path of the file in the database directory or the absolute path of files such as the backup file. If the target file does not exist or fails to be read, an error message is displayed.</p> <ul style="list-style-type: none"> If read_backup is false, the path format is <i>tablespace name/database oid/table relfilenode (physical file name)</i>. For example, base/16603/16394. If read_backup is true, path is a valid path. In this case, because other information about the input file cannot be obtained, you need to ensure that the input data is correct. <p>Note: Only U-page and UB-tree data pages can be edited and modified in page mode. Tables with tablespaces are not supported. Other information about the input file cannot be obtained. Therefore, you need to ensure that the input data type is correct.</p>
Input parameter	blocknum	bigint	<p>Block number of the page to be repaired. Value range: 0 to <i>MaxBlockNumber</i>.</p> <p>Reads the page corresponding to the specified physical or logical block number based on the read_backup field. If the specified block number is out of range, an error message is returned.</p>
Input parameter	offset	int	<p>In-page offset of the field to be modified. Value range: 0 to <i>BLCKSZ</i>.</p> <p>If the specified value is less than 0 or greater than that of BLCKSZ, the system view is used to return the corresponding error information.</p>
Input parameter	data	text	<p>Type of the target value to be modified. Type:</p> <ul style="list-style-type: none"> '0x': hexadecimal. '0b': binary. '0s': character string. <p>Others: If the value of the data parameter is not one of the preceding types, the data is a decimal character string.</p>

Category	Parameter	Type	Description
Input parameter	data_size	int	Length of the written data, in bytes. Value range: 1 to 8. If the specified write length is less than 1 byte or greater than 8 bytes, or the sum of offset and data_size is greater than the value of BLCKSZ , the system view is used to return the corresponding error information.
Input parameter	read_backup	bool	Specifies whether to read pages from the backup directory. If this parameter is set to false , the target page is read based on the logical block number. Otherwise, the page is read based on the physical block number.
Input parameter	storage_type	text	File storage mode. Currently, only the page storage mode is supported. This parameter is optional. <ul style="list-style-type: none"> 'page': page mode. 'segment': segment-page mode. This parameter is reserved and is not supported currently.
Output parameter	output_msg	text	If the modification is successful, the absolute path of the modified file is returned. The modified file is stored in the pg_log/dump directory. If the modification fails, a failure message is returned.

Note: In the example, transfer parameters based on the parameter description and use the actual physical path.

Example 1: Overwrite the data whose value is **0X1FFF** at the offset of 16 bytes on page 0 in the **base/15808/25075** table.

```
gaussdb=# select gs_edit_page_bypath('base/15808/25075',0,16,'0x1FFF', 2, false, 'page');
gs_edit_page_bypath
-----
/pg_log_dir/dump/1663_15808_25075_0.editpage
(1 rows)
```

Example 2: If the input parameter does not comply with the specifications, an error message is returned.

```
gaussdb=# select gs_edit_page_bypath('base/15808/25075', 0,16,'@1231!', 8, false, 'page');
gs_edit_page_bypath
-----
Error: the parameter 'data' decode failed.
(1 row)
```

Example 3: When the data to be written is the same as the original value, an alarm is returned.

```
gaussdb=# select gs_edit_page_bypath('/pg_log_dir/dump/1663_15808_25075_0.editpage',
0,16,'0x1FFF', 2, true, 'page');
gs_edit_page_bypath
-----
```

Warning: source buffer is consistent with target buffer.
(1 row)

- `gs_repair_page_bypath(src_path text, src_blkno int64, dest_path text, dest_blkno int64, storage_type text)`

Description: Specifies the path and page number of the source file, and writes the page to the specified page number of the target file. You can repair the pages of the primary node from the standby node. In addition, you can initialize bad blocks in this view. In a distributed environment, this function must be executed on DNs.

- a. The target page is overwritten and synchronized to the standby node. The page-based modification object supports the U-heap and UB-tree pages. The Undo Record page, Undo Slot page, compressed table, and Astore page will be supported later. System catalog files and data sections cannot be modified.
- b. With this function, you can overwrite target pages during the write operation. Before overwriting, the target page is backed up and flushed to a specified directory. The backup page can be written back to the target page. If an ordinary table is modified on the primary node, a new WAL is generated and synchronized to the standby node. If an ordinary table is modified on the standby node, no WAL is recorded.
- c. The repair view applies only to the primary node in a centralized or distributed system or the standby node when the read function is enabled on the standby node. Only the SYSADMIN or OPRADMIN in O&M mode can use this function. All modifications will be recorded in database logs. In addition, you are advised to enable the audit logging function of system functions before using this function to record audit information.
- d. The LSNs of the source and target pages must be the same. Otherwise, the repair fails.

Return type: text

 **CAUTION**

Calling this system function is a high-risk operation. Exercise caution when performing this operation.

Category	Parameter	Type	Description
Input parameter	src_path	text	Path of the source file. The following types of paths are supported: <ul style="list-style-type: none"> • Data files and index files: pg_log/dump/1663_15808_25075_0.editpage. • src_path is set to 'standby' on the primary node. That is, pages are read from the standby node to repair the primary node. • src_path is set to init_block on the primary node to allow skipping bad blocks in extreme scenarios.
Input parameter	src_blkno	bigint	Physical block number of the source page. Value range: 0 to <i>MaxBlockNumber</i> .
Input parameter	dest_path	text	Relative path of the target file. For example, base/15808/25075 .
Input parameter	dest_blkno	bigint	Logical block number of the target page. Value range: 0 to <i>MaxBlockNumber</i> .
Input parameter	storage_type	text	Storage mode of the target file. Currently, only the page storage mode is supported. This parameter is optional. <ul style="list-style-type: none"> • 'page': page mode. • 'segment': segment-page. This parameter is reserved and is not supported currently.
Output parameter	output_msg	text	If the write overwrite operation is successful, the backup path of the target page is returned. If the write overwrite operation fails, an error message is returned. The format of the flushed file name is <i>relfilepath_blocknum_timestamp.repairpage</i> .

Note: Transfer parameters based on the preceding table and ensure that the physical file exists. If the input parameter is abnormal or the restoration fails, an error is reported.

Example 1: Enter a file in a specified path to overwrite the target file.

```
gaussdb=# select * from gs_repair_page_bypath('pg_log/dump/1663_15991_16767_0.editpage', 0,
'base/15991/16767', 0, 'page');
          output_msg
-----
/pg_log_dir/dump/1663_15991_16767_0_738039702421788.repairpage
(1 row)
```

Example 2: Read pages from the standby node to repair the primary node.

```
gaussdb=# select * from gs_repair_page_bypath('standby', 0, 'base/15990/16768', 0, 'page');
          output_msg
-----
/pg_log_dir/dump/1663_15990_16768_0_738040397197907.repairpage
(1 row)
```

Example 3: Initialize the target page and skip bad blocks.

```
gaussdb=# select * from gs_repair_page_bypath('init_block', 0, 'base/15990/16768', 0, 'page');
          output_msg
-----
/pg_log_dir/dump/1663_15990_16768_0_738040768010281.repairpage
(1 row)
```

- `gs_repair_undo_byzone(zone_id int)`

Description: Transfers the ID of the undo zone to be repaired, repairs the metadata of the target undo zone, and returns the repair result details. If the undo zone is not repaired, no information is output.

Return type: record

Note: Currently, the function can be called only on the primary node. After the repair is successful, the repair will be synchronized to the standby node by recording Xlogs. The caller must be a system administrator or an O&M administrator in O&M mode. You are advised to enable the audit logging function before using the function to record audit information.

 **CAUTION**

Calling this system function is a high-risk operation. Exercise caution when performing this operation.

Table 7-133 `gs_repair_undo_byzone` parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID: <ul style="list-style-type: none"> • -1: repairs the metadata of all undo zones. • 0 to 1048575: repairs the metadata of the undo zone corresponding to the zone ID.
Output parameter	zone_id	int	Undo zone ID.

Category	Parameter	Type	Description
Output parameter	repair_detail	text	Repair result of the undo zone metadata corresponding to the zone ID. If the repair is successful, "rebuild undo meta succeed." is displayed. If the repair fails, "rebuild undo meta failed." as well as the failure cause is displayed.

Note: The output is one of the three cases based on the repair result.

Example 1: If the undo zone meta information corresponding to the entered **zone_id** is not damaged, no output is expected.

```
gaussdb=# select * from gs_repair_undo_byzone(4);
zone_id | repair_detail
-----+-----
(0 rows)
```

Example 2: If the undo zone metadata corresponding to the entered **zone_id** is successfully restored, the system displays a message indicating that the restoration is successful.

```
gaussdb=# select * from gs_repair_undo_byzone(78);
zone_id | repair_detail
-----+-----
      78 | rebuild undo meta succeed.
(1 row)
```

Example 3: If the undo zone metadata corresponding to the entered zone ID fails to be repaired, the detailed information about the repair failure is displayed.

```
gaussdb=# select * from gs_repair_undo_byzone(0);
zone_id | repair_detail
-----+-----
       0 | rebuild undo meta failed. try lock undo zone_id failed.
(1 row)
```

NOTE

If the undo zone to be repaired is damaged and the zone ID is occupied by another active thread, calling this repair function will end the thread that occupies the zone ID and forcibly repair the damaged undo zone metadata.

- `gs_verify_urq(index_oid oid, partindex_oid oid, blocknum bigint, queue_type text)`

Description: Verifies the correctness of the index recycling queue (potential queue/available queue/single page).

Parameter description: See [Table 7-134](#).

Return type: record

Table 7-134 gs_verify_urq parameters

Category	Parameter	Type	Description
Input parameter	index_oid	oid	UB-tree index OID. <ul style="list-style-type: none"> • Ordinary index: index OID. • Global index: GPI OID. • Local index: OID of the primary index.
Input parameter	partindex_oid	oid	UB-tree partitioned index OID. <ul style="list-style-type: none"> • Ordinary index: 0. • Global index: 0. • Local index: OID of the partitioned index (primary or secondary).
Input parameter	blocknum	bigint	Specifies the page number: <ul style="list-style-type: none"> • If the queue type is single page, the correctness of all tuples of blocknum on a single page is verified. The value range is [0, Queue file size/8192). • If the queue is empty or free, blocknum is an invalid value.
Input parameter	queue_type	text	Specifies the queue type: <ul style="list-style-type: none"> • empty queue: potential queue • free queue: available queue • single page: single-page queue
Output parameter	error_code	text	Error code
Output parameter	detail	text	Detailed error information and other key information.

Example 1: When using the example, transfer parameters based on the parameter description and use the actual OID and **blocknum**. Otherwise, an error is reported.

```
gaussdb=# select * from gs_verify_urq(16387, 0, 1, 'free queue');
error_code | detail
-----+-----
(0 rows)
```

Example 2: When using the example, transfer parameters based on the parameter description and use the actual OID and **blocknum**. Otherwise, an error is reported.

```
gaussdb=# select * from gs_verify_urq(16387, 0, 1, 'empty queue');
error_code |
```

```

detail
-----
+-----+
VERIFY_URQ_PAGE_ERROR | invalid urq meta: oid 16387, blkno 1, head_blkno = 1, tail_blkno = 3,
nblocks_upper = 4294967295, nblocks_lower = 1; urq_blocks = 6, index_blocks = 12
(1 row)
    
```

 **NOTE**

Currently, this API supports only Ustore index tables. If the verification of the index recycling queue is normal, the view does not display the error code and error details. Otherwise, the view displays the error code and error details. The error codes include "VERIFY_URQ_PAGE_ERROR", "VERIFY_URQ_LINK_ERROR", "VERIFY_URQ_HEAD_MISSED_ERROR", and "VERIFY_URQ_TAIL_MISSED_ERROR". If any of the preceding error codes is displayed, contact Huawei technical support to locate the fault.

- `gs_urq_dump_stat(index_oid oid, partindex_oid oid)`

Description: Queries information about a specified index recycling queue.

In the return result, **recentGlobalDataXmin** and **globalFrozenXid** are two oldestxmins used by the recycling queue to determine whether the index page can be recycled, **next_xid** is the XID of the next latest transaction, **urq_blocks** indicates the total number of pages in the recycling queue and information about valid pages in the free queue (available queue) and empty queue (potential queue).

Parameter description: See [Table 7-135](#).

Table 7-135 `gs_urq_dump_stat` parameters

Category	Parameter	Type	Description
Input parameter	<code>index_oid</code>	oid	UB-tree index OID. <ul style="list-style-type: none"> • Ordinary index: index OID. • Global index: GPI OID. • Local index: OID of the primary index.
Input parameter	<code>partindex_oid</code>	oid	UB-tree partitioned index OID. <ul style="list-style-type: none"> • Ordinary index: 0. • Global index: 0. • Local index: OID of the partitioned index (primary or secondary).
Output parameter	<code>result</code>	text	Detailed statistics about the index recycling queue.

Note: When using the example, transfer parameters based on the parameter description and use the actual OID. Otherwise, an error is reported.

Example:

```

gaussdb=# select * from gs_urq_dump_stat(16387, 0);
          result
-----
    
```

```
-----
urq stat info: recentGlobalDataXmin = 213156, globalFrozenXid = 213156, next_xid = 214157,
urq_blocks = 6,
free queue: head page blkno = 0 min_xid = 211187 max_xid = 214157, tail page blkno = 0
min_xid = 211187 max_xid = 214157,+
middle page min_xid = 1152921504606846975 max_xid = 0, valid_pages = 1, valid_items =
6, can_use_item = 3
empty queue: head page blkno = 1 min_xid = 212160 max_xid = 213160, tail page blkno = 3
min_xid = 213162 max_xid = 214156,+
middle page min_xid = 1152921504606846975 max_xid = 0, valid_pages = 2, valid_items =
999, can_use_item = 498
(1 row)
```

 **NOTE**

Currently, this API supports only Ustore index tables.

- `gs_repair_urq(index_oid oid, partindex_oid oid)`
Description: Repairs (with loss) index recycling queues (potential and available queues). The recycling queue file of the current index is deleted and an empty recycling queue file is created. If the repair is successful, "reinitial the recycle queue of index relation successfully" is displayed.

Parameter description: See [Table 7-136](#).

Note: The current function can be called only on the primary node.

Table 7-136 `gs_repair_urq` parameters

Category	Parameter	Type	Description
Input parameter	<code>index_oid</code>	oid	UB-tree index OID. <ul style="list-style-type: none"> • Ordinary index: index OID. • Global index: GPI OID. • Local index: OID of the primary index.
Input parameter	<code>partindex_oid</code>	oid	UB-tree partitioned index OID. <ul style="list-style-type: none"> • Ordinary index: 0. • Global index: 0. • Local index: OID of the partitioned index (primary or secondary).
Output parameter	<code>result</code>	text	If the repair is successful, "reinitial the recycle queue of index relation successfully" is displayed.

Example: When using the example, transfer parameters based on the parameter description and use the actual OID and **blocknum**. Otherwise, an error is reported.

```
gaussdb=# select * from gs_repair_urq(16387, 0);
result
```

reinitial the recycle queue of index relation successfully.
(1 row)

 **NOTE**

Currently, this API supports only Ustore index tables.

- `gs_get_standby_bad_block_info()`

Description: Displays the pages that have been detected on the standby node but have not been repaired. By default, only the initial user, users with the SYSADMIN permission, users with the OPRADMIN permission in the O&M mode, and users with the MONADMIN permission on the standby DN can view the information. Other users can view the information only after being granted permissions. There are four return values in the **invalid_type** column: **NOT_PRESENT** (the page does not exist), **NOT_INITIALIZED** (the page initialization fails), **LSN_CHECK_ERROR** (the LSN check fails), and **CRC_CHECK_ERROR** (the CRC check fails).

Return type: record

Example: If no page is detected but not repaired, no line is displayed.

```
gaussdb=# select * from gs_get_standby_bad_block_info();
 spc_node | db_node | rel_node | bucket_node | fork_num | block_num | invalid_type |
 master_page_lsn
-----+-----+-----+-----+-----+-----+-----+-----
 1663 | 16552 | 24745 | -1 | 0 | 0 | CRC_CHECK_ERROR | 0/B2009E8
(1 rows)
```

7.5.39 Functions of the XML Type

- `xmlparse ({ DOCUMENT | CONTENT } value [wellformed])`

Description: Generates XML values from character data.

Parameter: data of the TEXT type

Return type: XML

Example:

```
gaussdb=# SELECT XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual/<
title><chapter>...</chapter></book>');
 xmlparse
-----
<book><title>Manual</title><chapter>...</chapter></book>
(1 row)
gaussdb=# SELECT XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>');
 xmlparse
-----
abc<foo>bar</foo><bar>foo</bar>
(1 row)
gaussdb=# SELECT XMLPARSE (CONTENT 'abc<foo>bar</foo>' wellformed);
 xmlparse
-----
abc<foo>bar</foo>
(1 row)
```

- `xmlserialize({ DOCUMENT | CONTENT } value AS type)`

Description: Generates a string from an XML file.

Parameter: The type can be character, character varying, text, or any variant of them.

Return type: XML

Example:

```
gaussdb=# SELECT XMLSERIALIZE(CONTENT 'good' AS CHAR(10));
xmlserialize
```

```

-----
good
(1 row)
gaussdb=# SELECT xmlserialize(DOCUMENT '<head>bad</head>' as text);
xmlserialize
-----
<head>bad</head>
(1 row)

```

NOTE

If a string value is converted to XML without using the XMLPARSE or XMLSERIALIZE function, the **XML OPTION** session parameter determines the value: **DOCUMENT** or **CONTENT**. The **XML OPTION** session parameter can be set by the standard command.

```
SET XML OPTION { DOCUMENT | CONTENT };
```

Or use similar syntax to set this parameter.

```
SET xmloption TO { DOCUMENT | CONTENT };
```

- **xmlcomment(text)**

Description: Creates an XML value that contains an XML comment with the specified text as the content. The text does not contain the "--" character and does not end with a "-" character. Besides, the text should meet the format requirements of XML comments. If the parameter is empty, the result is also empty.

Parameter: data of the TEXT type

Return type: XML

Example:

```

gaussdb=# SELECT xmlcomment('hello');
xmlcomment
-----
<!--hello-->

```

- **xmlconcat(xml[, ...])**

Description: Concatenates a list of single XML values into a single value that contains an XML content fragment. Null values are ignored, and the result is null only when all parameters are null. In ORA-compatible mode, you can set **a_format_version** to **10c** and **a_format_dev_version** to **s2** to check whether the input segment is well-formed XML text.

Parameter: data of the XML type.

Return type: XML

Example 1:

```

gaussdb=# SET xmloption=content;
SET
gaussdb=# SELECT XMLCONCAT(('<?xml version="1.0" encoding="GB2312" standalone="no"?
><bar>foo</bar>'),('<?xml version="1.0" encoding="GB2312" standalone="no" ?><bar>foo</bar>'));
xmlconcat
-----
<?xml version="1.0" standalone="no"?><bar>foo</bar><bar>foo</bar>
(1 row)
gaussdb=# SELECT XMLCONCAT('abc');
xmlconcat
-----
abc>
(1 row)

```

Example 2: Syntax of the ORA-compatible database

```

gaussdb=# CREATE DATABASE gaussdb_ora DBCOMPATIBILITY='ORA';
CREATE DATABASE
gaussdb=# \c gaussdb_ora

```

```

gaussdb_ora=# SET a_format_version='10c';
SET
gaussdb_ora=# SET a_format_dev_version=s2;
SET
gaussdb_ora=# SET xmloption=content;
SET
gaussdb_ora=# SELECT XMLCONCAT(('<?xml version="1.0" encoding="GB2312" standalone="no"?
><bar>foo</bar>'),('<?xml version="1.0" encoding="GB2312" standalone="no" ?><bar>foo</bar>')) ;
          xmlconcat
-----
<?xml version="1.0" standalone="no"?><bar>foo</bar><bar>foo</bar>
(1 row)

gaussdb_ora=# SELECT XMLCONCAT('abc>');
ERROR:  invalid XML document
DETAIL:  line 1: Start tag expected, '<' not found
abc>
^
CONTEXT:  referenced column: xmlconcat
gaussdb_ora=# \c postgres
gaussdb=# DROP DATABASE gaussdb_ora;
DROP DATABASE

```

- `xmlagg(xml [order_by_clause])`

Description: Concatenates the input values called by the aggregate function. Cross-line concatenation is supported. For details about **order_by_clause**, see [SELECT](#). In ORA-compatible database mode, you can set **a_format_version** to **10c** and **a_format_dev_version** to **s2**. The **xmloption** parameter of the database is set to **content** by default. When **xmloption** is set to **document**, newline characters are used to concatenate multiple XML lines. If the encoding attribute value in the XML declaration is not the default UTF-8, the aggregation result contains the XML declaration.

Parameter: XML

Return type: XML

Example 1:

```

gaussdb=# CREATE TABLE xmltest (
          id int,
          data xml
        );
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:   Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
gaussdb=# INSERT INTO xmltest VALUES (1, '<value>one</value>');
INSERT 0 1
gaussdb=# INSERT INTO xmltest VALUES (2, '<value>two</value>');
INSERT 0 1
gaussdb=# SELECT xmlagg(data) FROM xmltest;
          xmlagg
-----
<value>one</value><value>two</value>
(1 row)

```

Example 2: Syntax of the ORA-compatible database

```

gaussdb=# SET xmloption=document;
SET
gaussdb=# SELECT xmlagg(data) FROM xmltest;
          xmlagg
-----
<value>one</value>+
<value>two</value>
(1 row)
gaussdb=# DELETE FROM XMLTEST;
DELETE 2
gaussdb=# INSERT INTO xmltest VALUES (1, '<?xml version="1.0" encoding="GBK"?><value>one</
value>');
INSERT 0 1

```

```

gaussdb=# INSERT INTO xmltest VALUES (2, '<?xml version="1.0" encoding="GBK"?><value>two</value>');
INSERT 0 1
gaussdb=# SELECT xmlagg(data) FROM xmltest;
          xmlagg
-----
<?xml version="1.0" encoding="GBK"?><value>one</value>+
<value>two</value>
(1 row)
gaussdb=# SELECT xmlagg(data order by id desc) FROM xmltest;
          xmlagg
-----
<?xml version="1.0" encoding="GBK"?><value>two</value>+
<value>one</value>
(1 row)

gaussdb=# DROP TABLE xmltest;

```

- xmlelement**([ENTITYESCAPING | NOENTITYESCAPING] { [NAME] element_name | EVALNAME element_name } [, xmlattributes([ENTITYESCAPING | NOENTITYESCAPING] value [[AS] attrname | AS EVALNAME attrname] [, ...])] [, content [[AS] alias] [, ...]])

Description: Generates an XML element with the given name, attribute, and content.

Return type: XML

Example:

```

gaussdb=# SELECT xmlelement(name foo);
xmlelement
-----
<foo/>

-- In ORA-compatible mode:
gaussdb=# CREATE DATABASE gaussdb_ora DBCOMPATIBILITY='ORA';
CREATE DATABASE
gaussdb=# \c gaussdb_ora
gaussdb_ora=# SET a_format_version='10c';
SET
gaussdb_ora=# SET a_format_dev_version=s2;
SET
-- If the keyword ENTITYESCAPING is not set in XMLElement by default or is set, the reserved
characters in the content of XMLElement are escaped.
gaussdb_ora=# SELECT xmlelement("entityescaping<>", 'a$<&"b');
          xmlelement
-----
<entityescaping<>>a$&gt;&lt;&amp;quot;b</entityescaping<>>
(1 row)

gaussdb_ora=# SELECT xmlelement(entityescaping "entityescaping<>", 'a$<&"b');
          xmlelement
-----
<entityescaping<>>a$&gt;&lt;&amp;quot;b</entityescaping<>>
(1 row)

-- When the keyword NOENTITYESCAPING is set in XMLElement, the reserved characters in the
content of XMLElement will not be escaped.
gaussdb_ora=# SELECT xmlelement(noentityescaping "entityescaping<>", 'a$<&"b');
          xmlelement
-----
<entityescaping<>>a$<&"b</entityescaping<>>
(1 row)

-- When [AS] alias is used to declare an alias for the content in XMLElement, the content value type
must be XML.
gaussdb_ora=# SELECT xmlelement("entityescaping<>", '<abc/>' b);
ERROR: argument of XMLELEMENT must be type xml, not type unknown

```

```

LINE 1: SELECT xmlelement("entityescaping<>", '<abc/>' b);
          ^
CONTEXT:  referenced column: xmlelement

gaussdb_ora=# SELECT xmlelement("entityescaping<>", '<abc/>' as b);
ERROR:  argument of XMLELEMENT must be type xml, not type unknown
LINE 1: SELECT xmlelement("entityescaping<>", '<abc/>' as b);
          ^
CONTEXT:  referenced column: xmlelement

gaussdb_ora=# SELECT xmlelement("entityescaping<>", xml('<abc/>') b);
          xmlelement
-----
<entityescaping<>><abc/></entityescaping<>>
(1 row)

gaussdb_ora=# SELECT xmlelement("entityescaping<>", xml('<abc/>') as b);
          xmlelement
-----
<entityescaping<>><abc/></entityescaping<>>
(1 row)

-- If the keyword ENTITYESCAPING is not set in XMLAttributes by default or is set, the reserved
characters in XMLAttributes are escaped.
gaussdb_ora=# SELECT xmlelement("entityescaping<>", xmlattributes('entityescaping<>'
"entityescaping<>"));
          xmlelement
-----
<entityescaping<> entityescaping<>="entityescaping&lt;&gt;"/>
(1 row)

gaussdb_ora=# SELECT xmlelement(name "entityescaping<>", xmlattributes(entityescaping
'entityescaping<>' "entityescaping<>"));
          xmlelement
-----
<entityescaping<> entityescaping<>="entityescaping&lt;&gt;"/>
(1 row)

-- When the NOENTITYESCAPING keyword is set in XMLAttributes, the reserved characters in
XMLAttributes will not be escaped.
gaussdb_ora=# SELECT xmlelement("entityescaping<>", xmlattributes(noentityescaping
'entityescaping<>' "entityescaping<>"));
          xmlelement
-----
<entityescaping<> entityescaping<>="entityescaping<>"/>
(1 row)

gaussdb_ora=# \c postgres
gaussdb=# DROP DATABASE gaussdb_ora;
DROP DATABASE

```

NOTE

1. For **xmlelement** and **xmlattributes**, when the value of **name** is **NULL**, the database behavior is different from that of the O database. When the **name** field of **xmlelement** is set to **NULL**, the name information is empty and the attribute information is not displayed. When the **name** field of **xmlattributes** is set to **NULL**, the attribute information is not displayed.
 2. After the following two parameters are set, the content escape rule of **xmlelement** is ORA compatible. If the two parameters are not set, the content escape rule of **xmlelement** is PG compatible.

```

SET a_format_version='10c';
SET a_format_dev_version=s2;

```
- **xmlforest(content [AS name] [, ...])**
Description: Generates an XML sequence of elements using the given name and content.

Return type: XML

Example:

```
gaussdb=# SELECT xmlforest('abc' AS foo, 123 AS bar);
xmlforest
-----
<foo>abc</foo><bar>123</bar>
```

- `xmlpi(name target [, content])`

Description: Creates an XML processing instruction. If the content is not empty, the content cannot contain character collations.

Return type: XML

Example:

```
gaussdb=# SELECT xmlpi(name php, 'echo "hello world";');
xmlpi
-----
<?php echo "hello world";?>
```

- `xmlroot(xml, version text | no value [, standalone yes|no|no value])`

Description: Modifies the attributes of the root node of an XML value. If a version is specified, it replaces the value in the version declaration of the root node. If a standalone property is specified, it replaces the value of standalone declaration in the root node.

Example:

```
gaussdb=# SELECT xmlroot('<?xml version="1.1"?><content>abc</content>',version '1.0', standalone yes);
xmlroot
-----
<?xml version="1.0" standalone="yes"?><content>abc</content>
(1 row)
```

- `xmlexists(text passing [BY REF] xml [BY REF])`

Description: Evaluates an XPath 1.0 expression (the first parameter) with the passed XML value as its context item. If the evaluation result generates an empty set of nodes, the function returns **false**. If any other value is generated, the function returns **true**. If the value of any parameter is null, the function returns **Null**. The non-null value passed as a context item must be an XML document, not a content fragment or any non-XML value.

Parameter: XML

Return type: Boolean

Example:

```
gaussdb=# SELECT xmlexists('/town[text() = "Toronto"]' PASSING BY REF '<towns><town>Toronto</town><town>Ottawa</town></towns>');
xmlexists
-----
t
(1 row)
```

- `xml_is_well_formed(text)`

Description: Checks whether the text is in the correct XML format and returns a Boolean value.

Parameter: text

Return type: Boolean

Example:

```
gaussdb=# SELECT xml_is_well_formed('<>');
xml_is_well_formed
-----
f
(1 row)
```

- `xml_is_well_formed_document(text)`
Description: Checks whether the text is in the correct XML format and returns a Boolean value.
Parameter: text
Return type: Boolean
Example:

```
gaussdb=# SELECT xml_is_well_formed_document('<pg:foo xmlns:pg="http://postgresql.org/stuff">bar</pg:foo>');
xml_is_well_formed_document
-----
t
(1 row)
```
- `xml_is_well_formed_content(text)`
Description: Checks whether the text is in the correct XML format and returns a Boolean value.
Parameter: text
Return type: Boolean
Example:

```
gaussdb=# SELECT xml_is_well_formed_content('k');
xml_is_well_formed_content
-----
t
(1 row)
```
- `xpath(xpath, xml [, nsarray])`
Description: Calculates an XPath 1.0 expression, for example, `xpath` (a text value), on XML data. It returns an array of XML values corresponding to the node set generated by the XPath expression. If the XPath expression returns a scalar value rather than a node set, a single-element array is returned.
The second parameter must be a well formed XML document. Note that it must have a single root node element.
The optional third parameter of the function is an array of namespace mappings. This array should be a two-dimensional text array with the length of the second axis being equal to 2 (that is, it should be an array of arrays, each of which consists of exactly 2 elements). The first element of each array entry is the namespace name (alias), and the second the namespace URI. It is not required that aliases provided in this array be the same as those being used in the XML document itself (that is, both in the XML document and in the XPath function context, aliases are local).
Return type: XML
Example:

```
gaussdb=# SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>', ARRAY[ARRAY['my', 'http://example.com']]);
xpath
-----
{test}
(1 row)
```
- `xpath_exists(xpath, xml [, nsarray])`
Description: This function is a special form of the `xpath` function. It returns a Boolean indicating whether the query is satisfied or not (that is, whether it produces any value other than an empty node set), instead of returning the individual XML values that satisfy the XPath 1.0 expression. This function is

equivalent to the standard XMLEXISTS predicate, but it also provides support for a namespace mapping parameter.

Return type: Boolean

Example:

```
gaussdb=# SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',ARRAY[ARRAY['my', 'http://example.com']]);
xpath_exists
-----
t
(1 row)
```

 **NOTE**

The following XML function examples show the data you need to prepare:

```
gaussdb=# CREATE SCHEMA testxmlschema;
CREATE SCHEMA
gaussdb=# CREATE TABLE testxmlschema.test1 (a int, b text);
CREATE TABLE
gaussdb=# INSERT INTO testxmlschema.test1 VALUES (1, 'one'), (2, 'two'), (-1, null);
INSERT 0 3
gaussdb=# CREATE DATABASE test;
CREATE DATABASE
```

-- After the example is executed, run the following commands to delete the preceding data:

```
gaussdb=# DROP DATABASE test;
DROP DATABASE
gaussdb=# DROP TABLE testxmlschema.test1;
DROP TABLE
gaussdb=# DROP SCHEMA testxmlschema;
DROP SCHEMA
```

- `query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)`

Description: This function maps the contents of a query to an XML schema document.

Return type: XML

Example:

```
gaussdb=# SELECT query_to_xml('SELECT * FROM testxmlschema.test1', false, false, '');
query_to_xml
-----
<table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
+
<row>+
+
+
<a>1</a>+
+
<b>one</b>+
+
</row>+
+
+
<row>+
+
+
<a>2</a>+
+
<b>two</b>+
+
</row>+
+
+
<row>+
+
+
<a>-1</a>+
+
</row>+
+
+
</table>+
(1 row)
```

- `query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)`

Description: This function maps the contents of a query to an XML document and an XML schema document, and joins the two documents together.

Return type: XML

Example:

```
gaussdb=# SELECT query_to_xmlschema('SELECT * FROM testxmlschema.test1', false, false, '');
               query_to_xmlschema
```

```
-----
<xsd:schema                                     +
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  +
  <xsd:simpleType name="INTEGER">               +
    <xsd:restriction base="xsd:int">           +
      <xsd:maxInclusive value="2147483647"/>   +
      <xsd:minInclusive value="-2147483648"/>  +
    </xsd:restriction>                         +
  </xsd:simpleType>                             +
  +
  <xsd:simpleType name="UDT.regression.pg_catalog.text"> +
    <xsd:restriction base="xsd:string">       +
      </xsd:restriction>                     +
  </xsd:simpleType>                             +
  +
  <xsd:complexType name="RowType">             +
    <xsd:sequence>                             +
      <xsd:element name="a" type="INTEGER" minOccurs="0"></xsd:element> +
      <xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></xsd:element>+
    </xsd:sequence>                           +
  </xsd:complexType>                           +
  +
  <xsd:complexType name="TableType">           +
    <xsd:sequence>                             +
      <xsd:element name="row" type="RowType" minOccurs="0" maxOccurs="unbounded"/> +
    </xsd:sequence>                           +
  </xsd:complexType>                           +
  +
  <xsd:element name="table" type="TableType"/> +
  +
</xsd:schema>
(1 row)
```

- query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)

Description: This function maps the contents of a query to an XML document and an XML schema document, and joins the two documents together.

Return type: XML

Example:

```
gaussdb=# SELECT query_to_xml_and_xmlschema('SELECT * FROM testxmlschema.test1', true, true, '');
               query_to_xml_and_xmlschema
```

```
-----
<xsd:schema                                     +
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  +
  <xsd:simpleType name="INTEGER">               +
    <xsd:restriction base="xsd:int">           +
      <xsd:maxInclusive value="2147483647"/>   +
      <xsd:minInclusive value="-2147483648"/>  +
    </xsd:restriction>                         +
  </xsd:simpleType>                             +
  +
  <xsd:simpleType name="UDT.regression.pg_catalog.text"> +
    <xsd:restriction base="xsd:string">       +
      </xsd:restriction>                     +
  </xsd:simpleType>                             +
  +
  <xsd:complexType name="RowType">             +
    <xsd:sequence>                             +
      <xsd:element name="a" type="INTEGER" nillable="true"></xsd:element> +
      <xsd:element name="b" type="UDT.regression.pg_catalog.text" nillable="true"></xsd:element>+
    </xsd:sequence>                           +
  </xsd:complexType>                           +
  +
  <xsd:element name="table" type="TableType"/> +
  +
</xsd:schema>
(1 row)
```

```

<xsd:element name="row" type="RowType"/>
</xsd:schema>
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <a>1</a>
  <b>one</b>
</row>
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <a>2</a>
  <b>two</b>
</row>
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <a>-1</a>
  <b xsi:nil="true"/>
</row>
(1 row)

```

- `cursor_to_xml(cursor refcursor, count int, nulls boolean,tableforest boolean, targetns text)`

Description: This function maps the contents of a cursor to an XML document.

Return type: XML

Example:

```

gaussdb=#CURSOR xc WITH HOLD FOR SELECT * FROM testxmlschema.test1 ORDER BY 1, 2;
DECLARE CURSOR
gaussdb=# SELECT cursor_to_xml('xc':refcursor, 5, false, true, '');
          cursor_to_xml

```

```

-----
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
  <a>-1</a>          +
</row>              +
  <row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
  <a>1</a>          +
  <b>one</b>        +
</row>              +
  <row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
  <a>2</a>          +
  <b>two</b>        +
</row>              +
(1 row)

```

- `cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)`

Description: This function maps the contents of a cursor to an XML schema document.

Return type: XML

Example:

```

gaussdb=# SELECT cursor_to_xmlschema('xc':refcursor, true, false, '');
          cursor_to_xmlschema

```

```

<xsd:schema                                +
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  +
  <xsd:simpleType name="INTEGER">          +
    <xsd:restriction base="xsd:int">      +
      <xsd:maxInclusive value="2147483647"/>
      +
      <xsd:minInclusive value="-2147483648"/>
      +
    </xsd:restriction>
    +
  </xsd:simpleType>
  +
  <xsd:simpleType name="UDT.regression.pg_catalog.text">
    <xsd:restriction base="xsd:string">
      <xsd:restriction base="xsd:string">
        +
      </xsd:restriction>
    </xsd:restriction>
  </xsd:simpleType>
  +
  <xsd:complexType name="RowType">
    <xsd:sequence>
      <xsd:element name="a" type="INTEGER" nillable="true"></xsd:element>
      +
      <xsd:element name="b" type="UDT.regression.pg_catalog.text" nillable="true"></xsd:element>+
    </xsd:sequence>
    +
  </xsd:complexType>
  +
  <xsd:complexType name="TableType">
    <xsd:sequence>
      <xsd:element name="row" type="RowType" minOccurs="0" maxOccurs="unbounded"/>
      +
    </xsd:sequence>
    +
  </xsd:complexType>
  +
  <xsd:element name="table" type="TableType"/>
  +
</xsd:schema>
(1 row)

```

- schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)

Description: This function maps the contents of the entire schema to an XML document.

Return type: XML

Example:

```

gaussdb=# SELECT schema_to_xml('testxmlschema', false, true, '');
          schema_to_xml

```

```

-----
<testxmlschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
  +
  <test1>
    +
    <a>1</a>
    <b>one</b>
    +
  </test1>
  +
  <test1>
    +
    <a>2</a>
    <b>two</b>
    +
  </test1>
  +
  <test1>
    +
    <a>-1</a>
    +
  </test1>
  +
</testxmlschema>
(1 row)

```

- schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)

Description: This function maps the contents of the entire schema to an XML schema document.

Return type: XML

Example:

```
gaussdb=# SELECT schema_to_xmlschema('testxmlschema', false, true, '');
                schema_to_xmlschema
-----
<xsd:schema                                +
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                                +
  <xsd:simpleType name="INTEGER">
    <xsd:restriction base="xsd:int">
      <xsd:maxInclusive value="2147483647"/>
      <xsd:minInclusive value="-2147483648"/>
    </xsd:restriction>
  </xsd:simpleType>
                                +
  <xsd:simpleType name="UDT.t1.pg_catalog.text">
    <xsd:restriction base="xsd:string">
    </xsd:restriction>
  </xsd:simpleType>
                                +
  <xsd:complexType name="SchemaType.t1.testxmlschema">
    <xsd:sequence>
      <xsd:element name="test1" type="RowType.t1.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
    </xsd:sequence>
  </xsd:complexType>
                                +
  <xsd:element name="testxmlschema" type="SchemaType.t1.testxmlschema"/>
+
                                +
</xsd:schema>
(1 row)
```

- schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)

Description: This function maps the contents of the entire schema to an XML document and an XML schema document, and joins the two documents together.

Return type: XML

Example:

```
gaussdb=# SELECT schema_to_xml_and_xmlschema('testxmlschema', true, true, 'foo');
                schema_to_xml_and_xmlschema
-----
<testxmlschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="foo"
xsi:schemaLocation="foo #">+
                                +
  <xsd:schema                                +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="foo"
    elementFormDefault="qualified">
                                +
  <xsd:simpleType name="INTEGER">
    <xsd:restriction base="xsd:int">
      <xsd:maxInclusive value="2147483647"/>
      <xsd:minInclusive value="-2147483648"/>
    </xsd:restriction>
  </xsd:simpleType>
                                +
  <xsd:simpleType name="UDT.t1.pg_catalog.text">
    <xsd:restriction base="xsd:string">
    </xsd:restriction>
                                +
```

```

</xsd:simpleType>
+
+
<xsd:complexType name="SchemaType.t1.testxmlschema">
+
+
<xsd:sequence>
+
+
<xsd:element name="test1" type="RowType.t1.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/> +
</xsd:sequence>
+
</xsd:complexType>
+
<xsd:element name="testxmlschema"
type="SchemaType.t1.testxmlschema"/>
+
</xsd:schema>
+
+
<test1>
+
+
<a>1</a>
+
<b>one</b>
+
</test1>
+
<test1>
+
+
<a>2</a>
+
<b>two</b>
+
</test1>
+
<test1>
+
+
<a>-1</a>
+
<b xsi:nil="true"/>
+
</test1>
+
+
</testxmlschema>
+
(1 row)

```

- database_to_xml(nulls boolean, tableforest boolean, targets text)

Description: This function maps the contents of the entire database to an XML document.

Return type: XML

Example:

```

gaussdb=# SELECT database_to_xml(true, true, 'test');
database_to_xml
-----
<test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="test">+
+
<dbe_x005F_xml>
+
+
</dbe_x005F_xml>
+
+
<dbe_x005F_xmldom>
+
+
</dbe_x005F_xmldom>
+
+
<dbe_x005F_xmlparser>
+
+
</dbe_x005F_xmlparser>
+
+
<public>
+
+
</public>
+
+
</test>
+
(1 row)

```



```

<dbe_x005F_xmldom>
+
+
</dbe_x005F_xmldom>
+
+
<dbe_x005F_xmlparser>
+
+
</dbe_x005F_xmlparser>
+
+
<public>
+
+
</public>
+
+
</test>
+
(1 row)

```

- **table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)**
Description: This function maps the contents of a relational table to an XML document.

Return type: XML

Example:

```

gaussdb=# SELECT table_to_xml('testxmlschema.test1', false, false, '');
table_to_xml

```

```

-----
<test1 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
+
<row>
+
+
+
<a>1</a>
+
+
<b>one</b>
+
</row>
+
+
<row>
+
+
+
<a>2</a>
+
+
<b>two</b>
+
</row>
+
+
<row>
+
+
+
<a>-1</a>
+
+
</row>
+
+
</test1>
+
(1 row)

```

- **table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)**

Description: This function maps the contents of a relational table to an XML schema document.

Return type: XML

Example:

```

gaussdb=# SELECT table_to_xmlschema('testxmlschema.test1', false, false, '');
table_to_xmlschema

```

```

-----
<xsd:schema
+
+
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
+
+
<xsd:simpleType name="INTEGER">
+
+
+
<xsd:restriction base="xsd:int">
+
+
+
<xsd:maxInclusive value="2147483647"/>
+
+
<xsd:minInclusive value="-2147483648"/>
+
+
</xsd:restriction>
+
+
</xsd:simpleType>
+
+
<xsd:simpleType name="UDT.regression.pg_catalog.text">
+
(1 row)

```

```

<xsd:restriction base="xsd:string">
</xsd:restriction>
</xsd:simpleType>
+
+
+
<xsd:complexType name="RowType.regression.testxmlschema.test1">
+
<xsd:sequence>
+
<xsd:element name="a" type="INTEGER" minOccurs="0"></
xsd:element>
+
<xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></
xsd:element>
+
</xsd:sequence>
+
</xsd:complexType>
+
+
<xsd:complexType name="TableType.regression.testxmlschema.test1">
+
<xsd:sequence>
+
<xsd:element name="row" type="RowType.regression.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
</xsd:sequence>
+
</xsd:complexType>
+
+
<xsd:element name="test1"
type="TableType.regression.testxmlschema.test1"/>
+
+
</xsd:schema>
(1 row)

```

- `table_to_xml_and_xmlschema`(tbl regclass, nulls boolean, tableforest boolean, targetns text)

Description: This function maps the contents of a relational table to an XML document and an XML schema document, and joins the two documents together.

Return type: XML

Example:

```
gaussdb=# SELECT table_to_xml_and_xmlschema('testxmlschema.test1', false, false, '');
table_to_xml_and_xmlschema
```

```

-----
<test1 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="#">
+
+
+
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
+
+
+
<xsd:simpleType name="INTEGER">
<xsd:restriction base="xsd:int">
+
<xsd:maxInclusive value="2147483647"/>
+
<xsd:minInclusive value="-2147483648"/>
+
</xsd:restriction>
+
</xsd:simpleType>
+
+
<xsd:simpleType name="UDT.regression.pg_catalog.text">
<xsd:restriction base="xsd:string">
+
</xsd:restriction>
+
</xsd:simpleType>
+
+
<xsd:complexType name="RowType.regression.testxmlschema.test1">
+
<xsd:sequence>
+
<xsd:element name="a" type="INTEGER" minOccurs="0"></
xsd:element>
+
<xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></
xsd:element>
+
</xsd:sequence>
+
</xsd:complexType>
+
+
<xsd:complexType name="TableType.regression.testxmlschema.test1">

```

```

+
<xsd:sequence> +
  <xsd:element name="row" type="RowType.regression.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
</xsd:sequence> +
</xsd:complexType> +
+
<xsd:element name="test1" +
type="TableType.regression.testxmlschema.test1"/> +
+
</xsd:schema> +
+
<row> +
  <a>1</a> +
  <b>one</b> +
</row> +
+
<row> +
  <a>2</a> +
  <b>two</b> +
</row> +
+
<row> +
  <a>-1</a> +
</row> +
+
</test1> +
(1 row)

```

 **NOTE**

- For XPath-related functions, only XPath() and XPath_exists() are supported. These functions use the XPath language to query XML files and depend on the Libxml2 library, which is provided only in XPath1.0. Therefore, only XPath 1.0 is supported.
- The XQuery, XML extension, and XSLT functions are not supported.

7.5.40 Functions of the XMLType Type

- createxml(varchar2[,varchar2,numeric ,numeric])

Description: Statically creates the XMLType type. The input parameters are of the varchar2 type.

Parameters: The first parameter is the character string to be converted to XMLType (required column). The second parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The third parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is 0 by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is well-formed (optional column, which is 0 by default and does not take effect currently).

Return type: XMLType

Example:

```

gaussdb=# SELECT createxml('<a>123</a>');
createxml
-----
<a>123</a>
(1 row)

```

 NOTE

- Different from that in database A, in PL/SQL, createxml allows input parameters to be empty strings and returns **NULL**.
- For character encoding, only UTF-8, GBK, and LATIN1 to LATIN10 are supported, and the **version** field can only be set to **1.x**.
- The createxml function can be called using the xmltype.createxml() syntax.

Example:

```
gaussdb=# SELECT xmltype.createxml('<a>123</a>');
createxml
-----
<a>123</a>
(1 row)
```

- In this chapter, the function whose input parameter is xmltype() can be called in xmltype().func() mode. The XMLType type returned by a function is transferred to the next function as the input parameter. This syntax supports multi-layer nesting. (If the input parameter is defined as XMLType by a user, this syntax is not supported.)

Example:

```
gaussdb=# select xmltype('<a>123<b>456</b></a>').extract('/a/b').getstringval();
xmltypefunc
-----
<b>456</b>
(1 row)
```

The actual effect of the preceding example is the same as that of the following function nesting:

```
gaussdb=# select getstringval(extractxml(xmltype('<a>123<b>456</b></a>'),'a/b'));
getstringval
-----
<b>456</b>
(1 row)
```

- In a stored procedure, variables of the XMLType type can call functions in a.func() mode. This syntax supports one-layer nesting.

Example:

```
gaussdb=# declare
a xmltype;
b varchar2;
begin
a:=xmltype('<a>123<b>456</b></a>');
b:=a.getstringval();
RAISE NOTICE 'xmltype_str is : %',b;
end;
/
NOTICE: xmltype_str is : <a>123<b>456</b></a>
```

- createxml(clob [,varchar2,numeric ,numeric])

Description: Statically creates the XMLType type. The input parameters are of the CLOB type.

Parameters: The first parameter is the CLOB to be converted to XMLType (required column). The second parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The third parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

Return type: XMLType

Example:

```
gaussdb=# declare
xmltype_clob clob;
xmltype_obj xmltype;
xmltype_str varchar2(1000);
begin
xmltype_clob := '<a>123</a>';
xmltype_obj := createxml(xmltype_clob);
xmltype_str := xmltype_obj.getstringval();
RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE: xmltype_str is : <a>123</a>
```

 **NOTE**

The maximum size of the input parameter of the CLOB type is 1 GB minus 1 byte.

- `createxml(blob, numeric [,varchar2,numeric ,numeric])`

Description: Statically creates the XMLType type. The input parameters are of the BLOB type.

Parameters: The first parameter is the BLOB to be converted to XMLType (required column). The second parameter is the character set ID of the input XML data (required column). The third parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fifth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

Return type: XMLType

Example:

```
gaussdb=# declare
xmltype_blob blob;
xmltype_obj xmltype;
xmltype_str varchar2(1000);
begin
xmltype_blob := xmltype('<a>123</a>').getblobval(7);
xmltype_obj := createxml(xmltype_blob,7);
xmltype_str := xmltype_obj.getstringval();
RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE: xmltype_str is : <?xml version="1.0" encoding="UTF8"?>
<a>123</a>
```

 **NOTE**

- The maximum size of input parameters of the BLOB type is 256 MB minus 1 byte.
- The character set ID ranges from 1 to 41.
- `getblobval(xmltype, numeric)`

Description: Converts the XMLType type to the BLOB type. The `xmltype().func()` method can be called.

Parameters: The first parameter is of the XMLType type, and the second parameter is the ID of the target character set to be converted.

Return type: BLOB

Example:

```
gaussdb=# SELECT getblobval(xmltype('<asd/>'),7);
          getblobval
-----
3C3F786D6C2076657273696F6E3D22312E302220656E636F64696E673D2255544638223F3E0A3C6173
642F3E
(1 row)
```

xmltype ().func ():

```
gaussdb=# select xmltype('<asd/>').getblobVal(7);
          xmltypefunc
-----
3C3F786D6C2076657273696F6E3D22312E302220656E636F64696E673D2255544638223F3E0A3C6173
642F3E
(1 row)
```

 **NOTE**

The maximum length of the input parameter of the XMLType type is 256 MB minus 1 byte.

- **getclobval(xmltype)**

Description: Converts the XMLType type to the CLOB type. The xmltype().func() method can be called.

Parameter: The input parameter is of the XMLType type.

Return type: CLOB

Example:

```
gaussdb=# SELECT getclobval(xmltype('<a>123</a>'));
          getclobval
-----
<a>123</a>
(1 row)
```

xmltype ().func ():

```
gaussdb=# SELECT xmltype('<a>123</a>').getclobval();
          xmltypefunc
-----
<a>123</a>
(1 row)
```

- **getnumberval(xmltype)**

Description: Converts the XMLType type to the numeric type. The xmltype().func() method can be called.

Parameter: The input parameter is of the XMLType type.

Return type: numeric

Example:

```
gaussdb=# SELECT getnumberval(xmltype('<a>123</a>').extract('/a/text()));
          getnumberval
-----
123
(1 row)
```

xmltype ().func ():

```
gaussdb=# SELECT xmltype('<a>123</a>').extract('/a/text()').getnumberval();
          xmltypefunc
-----
123
(1 row)
```

- **isfragment(xmltype)**

Description: Returns a result indicating whether the XMLType type is fragment (1) or document (0). The xmltype().func() method can be called.

Parameter: The input parameter is of the XMLType type.

Return type: numeric

Example:

```
gaussdb=# SELECT isfragment(xmltype('<a>123</a>');
isfragment
```

```
-----
0
(1 row)
```

xmltype ().func ():

```
gaussdb=# SELECT xmltype('<a>123</a>').isfragment();
xmltypefunc
```

```
-----
0
(1 row)
```

- xmltype(varchar2[,varchar2,numeric ,numeric])

Description: Creates the XMLType type from the varchar2 type.

Parameters: The first parameter is the character string to be converted to XMLType (required column). The second parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The third parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

Return type: XMLType

Example:

```
gaussdb=# SELECT xmltype('<a>123</a>');
xmltype
```

```
-----
<a>123</a>
(1 row)
```

NOTE

- Different from that in database A, in PL/SQL, XMLType allows input parameters to be empty strings and returns **NULL**.
- For character encoding, only UTF-8, GBK, and LATIN1 to LATIN10 are supported, and the **version** field can only be set to **1.x**.
- xmltype(clob [,varchar2,numeric ,numeric])

Description: Creates the XMLType type from the CLOB type.

Parameters: The first parameter is the CLOB to be converted to XMLType (required column). The second parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The third parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

Return type: XMLType

Example:

```
gaussdb=# declare
xmltype_clob clob;
```

```
xmltype_obj xmltype;
xmltype_str varchar2(1000);
begin
xmltype_clob := '<a>123</a>';
xmltype_obj := xmltype(xmltype_clob);
xmltype_str := xmltype_obj.getstringval();
RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE: xmltype_str is : <a>123</a>
```

NOTE

The maximum size of the input parameter of the CLOB type is 1 GB minus 1 byte.

- `xmltype(blob, numeric [,varchar2,numeric ,numeric])`

Description: Creates the XMLType type from the BLOB type.

Parameters: The first parameter is the BLOB to be converted to XMLType (required column). The second parameter is the character set ID of the input XML data. The third parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fifth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

Return type: XMLType

Example:

```
gaussdb=# declare
xmltype_blob blob;
xmltype_obj xmltype;
xmltype_str varchar2(1000);
begin
xmltype_blob := getblobval(createxml('<a>123</a>'),7);
xmltype_obj := xmltype(xmltype_blob,7);
xmltype_str := xmltype_obj.getstringval();
RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE: xmltype_str is : <?xml version="1.0" encoding="UTF8"?>
<a>123</a>
```

NOTE

- The maximum size of input parameters of the BLOB type is 256 MB minus 1 byte.
- The character set ID ranges from 1 to 41.
- `getstringval(xmltype)`

Description: Converts the XMLType to a string.

Parameter: XMLType to be converted.

Return type: varchar2

The getstringval function can be called in either of the following ways:

Example 1:

```
gaussdb=# SELECT getstringval('<a>123<b>456</b></a>');
getstringval
-----
<a>123<b>456</b></a>
(1 row)
```

Example 2: The calling mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').getstringval();
xmltypefunc
-----
<a>123<b>456</b></a>
(1 row)
```

- **getrootelement(xmltype)**

Description: Gets the root element of the XMLType type.

Parameter: XMLType whose root element is to be obtained.

Return type: varchar2

The getrootelement function can be called in either of the following ways:

Example 1:

```
gaussdb=# SELECT getrootelement('<a>123<b>456</b></a>');
getrootelement
-----
a
(1 row)
```

Example 2: The calling mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').getrootelement();
xmltypefunc
-----
a
(1 row)
```

- **getnamespace(xmltype)**

Description: Gets the namespace of the XMLType top-level element.

Parameter: XMLType whose namespace is to be obtained.

Return type: varchar2

The getnamespace function can be called in either of the following ways:

Example 1:

```
gaussdb=# SELECT getnamespace('<c:a xmlns:c="asd">123<d:b xmlns:d="qwe">456</d:b></c:a>');
getnamespace
-----
asd
(1 row)
```

Example 2: The calling mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<c:a xmlns:c="asd">123<d:b xmlns:d="qwe">456</d:b></c:a>').getnamespace();
xmltypefunc
-----
asd
(1 row)
```

- **existsnode(xmltype, varchar2[, varchar2])**

Description: Determines whether the XML node exists in XMLType based on the XPath expression. If the XML node exists, **1** is returned. Otherwise, **0** is returned.

Parameters: XMLType to be queried, path of the XPath node to be queried, and namespace of the XPath path (If the input parameter has a namespace, aliases must be defined for both the XPath and namespace, as shown in example 3.)

Return type: numeric

The existsnode function can be called in either of the following ways:

Example 1:

```
gaussdb=# SELECT existsnode('<a>123<b>456</b></a>', '/a/b');
existsnode
-----
          1
(1 row)
```

Example 2: The calling mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').existsnode('/a/b');
xmltypefunc
-----
          1
(1 row)
```

Example 3:

```
gaussdb=# SELECT existsnode('<a:b xmlns:a="asd">123<c>456</c></a:b>', '/a:b/c', 'xmlns:a="asd"');
existsnode
-----
          1
(1 row)
```

Example 4: The calling mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').existsnode('/a:b/c', 'xmlns:a="asd"');
xmltypefunc
-----
          1
(1 row)
```

- `extractxml(xmltype, varchar2[, varchar2])`

Description: Checks whether an XML node exists in the given XMLType based on the XPath expression. If yes, the XMLType containing the node is returned. If no, **NULL** is returned. The return value can be inserted into a table of the XMLType type.

Parameters: XMLType to be queried, path of the XPath node to be queried, and namespace of the XPath path (If the input parameter has a namespace, aliases must be defined for both the XPath and namespace, as shown in example 3.)

Return type: XMLType

The extractxml function can be called in either of the following ways:

Example 1:

```
gaussdb=# SELECT extractxml('<a>123<b>456</b></a>', '/a/b');
extractxml
-----
<b>456</b>
(1 row)
```

Example 2: The calling mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').extract('/a/b');
xmltypefunc
-----
<b>456</b>
(1 row)
```

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').extractxml('/a/b');
xmltypefunc
-----
<b>456</b>
(1 row)
```

Example 3:

```
gaussdb=# SELECT extractxml('<a:b xmlns:a="asd">123<c>456</c></a:b>', '/a:b', 'xmlns:a="asd"');
extractxml
-----
```

```
<a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)
```

Example 4: The calling mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').extract('/a:b','xmlns:a="asd"');
xmltypefunc
```

```
-----
<a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)
```

```
gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').extractxml('/
a:b','xmlns:a="asd"');
xmltypefunc
```

```
-----
<a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)
```

7.5.41 Other System Functions

Other system functions are classified into PG-compatible functions and internal functions. These functions are not recommended. If you need to use them, contact Huawei technical support.

PG-compatible Functions and Operators

The following table lists the built-in functions and operators of GaussDB that are compatible with PG. Some functions (such as send and rcv) are internal functions, and users cannot use them. Other functions are not recommended. If you need to use them, contact Huawei technical support.

_pg_char_max_length	_pg_char_octet_length	_pg_datetime_precision	_pg_expararray	_pg_index_position	_pg_interval_type	_pg_numeric_precision
_pg_numeric_radix	_pg_numeric_scale	_pg_truetypid	_pg_truetypm od	q	abs	abstime
abstimeeq	abstimege	abstimegt	abstimein	abstimele	abstimeelt	abstimene
abstimeout	abstimercv	abstimesend	aclcontains	acldefault	aclexplode	aclinsert
acliteq	aclitein	acliteout	aclremove	acos	age	akeys
any_in	any_out	anyarray_in	anyarray_out	anyarray_rcv	anyarray_send	anyelement_in
anyelement_out	anyenum_in	anyenum_out	anynonarray_in	anynonarray_out	anyrange_in	anyrange_out
anytextcat	area	areajoinse	areaset	array_agg	array_agg_finalfn	array_agg_transfn

array_append	array_cat	array_dims	array_eq	array_fill	array_ge	array_gt
array_in	array_larger	array_length	array_length	array_lower	array_lt	array_ndims
array_ne	array_out	array_prepend	array_recv	array_send	array_smaller	array_to_json
array_to_string	array_type_analyze	array_upper	array_contains	array_contains	array_concatjoin	array_contsel
arrayoverlap	ascii	asin	atan	atan2	avals	avg
big5_to_euc_tw	big5_to_mic	big5_to_utf8	bit	bit_and	bit_in	bit_length
bit_or	bit_out	bit_recv	bit_send	bitand	bitcat	bitcmp
biteq	bitge	bitgt	bitle	bitlt	bitne	bitnot
bitor	bitshiftleft	bitshiftright	bittypmodin	bittypmodout	bitxor	bool
bool_and	bool_or	booland_statefunc	booleq	boolge	boolgt	boolin
boolle	boollt	boolne	boolor_statefunc	boolout	boolrecv	boolsend
box	box_above	box_above_eq	box_adj	box_below	box_below_eq	box_center
box_contain	box_contain_pt	box_contained	box_distance	box_div	box_eq	box_ge
box_gt	box_in	box_intersect	box_le	box_left	box_lt	box_mul
box_out	box_overabove	box_overbelow	box_overlap	box_overleft	box_overright	box_recv
box_right	box_same	box_send	box_sub	bpchar	bpchar_larger	bpchar_pattern_ge
bpchar_pattern_gt	bpchar_pattern_le	bpchar_pattern_lt	bpchar_smaller	bpchar_sortsupport	bpcharcmp	bpchareq

bpcharge	bpchargt	bpchariclike	bpcharicnlike	bpcharicregexeq	bpcharicregexne	bpcharin
bpcharle	bpcharlike	bpcharlt	bpcharne	bpcharnlike	bpcharout	bpcharrecv
bpcharregexeq	bpcharregexne	bpcharend	bpchartypmodin	bpchartypmodout	broadcast	btabstimecmp
btarraycmp	btbeginscan	btboolcmp	btbpcchar_pattern_cmp	btbuild	btbuildempty	btbulkdelete
btcanreturn	btcharcmp	btcostestimate	btendscan	btfloat48cmp	btfloat4cmp	btfloat4sortsupport
btfloat84cmp	btfloat8cmp	btfloat8sortsupport	btgetbitmap	btgettuple	btinsert	btint24cmp
btint28cmp	btint2cmp	btint2sortsupport	btint42cmp	btint48cmp	btint4cmp	btint4sortsupport
btint82cmp	btint84cmp	btint8cmp	btint8sortsupport	btmarkpos	btnamecmp	btnamesortsupport
btoidcmp	btoidsortsupport	btoidvectorcmp	btoptions	btrecordcmp	btreltimecmp	btrescan
btrestrpos	btrim	bttext_pattern_cmp	bttextcmp	bttextsortsupport	bttidcmp	btintervalcmp
btvacuumcleanup	bytea_sortsupport	bytea_string_agg_fn	bytea_string_agg_transfn	byteacat	byteacmp	byteaeq
byteage	byteagt	byteain	byteale	bytealike	bytealt	byteane
byteanlike	byteaout	bytearecv	byteasend	cash_cmp	cash_div_cash	cash_div_float4
cash_div_float8	cash_div_int2	cash_div_int4	cash_div_int8	cash_eq	cash_ge	cash_gt
cash_in	cash_le	cash_lt	cash_mi	cash_mul_float4	cash_mul_float8	cash_mul_int2

cash_mul_int4	cash_mul_int8	cash_ne	cash_out	cash_pl	cash_recv	cash_send
cashlarger	cashsmaller	cbrt	ceil	ceiling	center	char
char_length	character_length	chareq	charge	charget	charin	charle
charlt	charne	charout	charrecv	charsend	chr	cideq
cidin	cidout	cidr	cidr_in	cidr_out	cidr_recv	cidr_send
cidrecv	cidsend	circle	circle_above	circle_add_pt	circle_below	circle_center
circle_contain	circle_contain_pt	circle_contained	circle_distance	circle_div_pt	circle_eq	circle_ge
circle_gt	circle_in	circle_le	circle_left	circle_lt	circle_mul_pt	circle_ne
circle_out	circle_overabove	circle_overbelow	circle_overlap	circle_overleft	circle_outright	circle_recv
circle_right	circle_same	circle_sen	circle_sub_pt	clock_timestamp	close_lb	close_ls
close_lseg	close_pb	close_pl	close_ps	close_sb	close_sl	col_description
concat	concat_ws	contjoinsel	contsel	convert	convert_from	convert_to
corr	cos	cot	count	covar_pop	covar_samp	cstring_in
cstring_out	cstring_recv	cstring_send	cume_dist	current_database	current_query	current_schema
xpath_exists	current_setting	current_user	currtid	currtid2	curval	cursor_to_xml
cursor_to_xmlschema	database_to_xml	database_to_xml_and_xmlschema	database_to_xmlschema	date	date_cmp	date_cmp_timestamp
date_cmp_timestamptz	date_eq	date_eq_timestamp	date_eq_timestampz	date_ge	date_ge_timestamp	date_ge_timestamptz

date_gt	date_gt_timestam p	date_gt_timestam ptz	date_in	date_lar ger	date_l e	date_le_tim estamp
date_le_tim estamptz	date_lt	date_lt_timestam p	date_lt_timest amptz	date_mi	date_mi_int erval	date_mii
date_ne	date_ne_timesta mp	date_ne_timestam ptz	date_o ut	date_pl interval	date_p li	date_recv
date_send	date_sm aller	date_sort support	datera nge_ca nonical	dateran ge_subd iff	dateti me_pl	datetimetz_ pl
dcbrt	decode	defined	degree s	delete	dense_ rank	dexp
diagonal	diameter	dispell_ini t	dispell_ lexize	dist_cpo ly	dist_lb	dist_pb
dist_pc	dist_pl	dist_ppat h	dist_ps	dist_sb	dist_sl	div
dlog1	dlog10	domain_i n	domai n_recv	dpow	droun d	dsimple_init
dsimple_lexi ze	dsnowba ll_init	dsnowbal l_lexize	dsqrt	dsynony m_init	dsynony m_l exize	dtrunc
each	enum_ne	enum_ou t	enum_ range	enum_r ecv	enum_ send	enum_small er
eqjoinsel	eqsel	euc_cn_to _mic	euc_cn _to_utf 8	euc_jis_ 2004_to _shift_jis _2004	euc_jis _2004_ to_utf 8	euc_jp_to_m ic
euc_jp_to_sji s	euc_jp_to _utf8	euc_kr_to _mic	euc_kr_ to_utf8	euc_tw_ to_big5	euc_t w_to_ mic	euc_tw_to_u tf8
every	exist	exists_all	exists_ any	exp	factori al	family
fdw_handler _in	fdw_han dler_out	fetchval	first_va lue	float4	float4_ accum	float48div
float48eq	float48g e	float48gt	float48 le	float48l t	float4 8mi	float48mul
float48ne	float48pl	float4abs	float4d iv	float4eq	float4 ge	float4gt

float4in	float4larger	float4le	float4lt	float4mi	float4mul	float4ne
float4out	float4pl	float4recv	float4send	float4smaller	float4um	float4up
float8	float8_accum	float8_avg	float8_collect	float8_corr	float8_covar_pop	float8_covar_samp
float8_regr_accum	float8_regr_avgx	float8_regr_avgy	float8_regr_collect	float8_regr_intercept	float8_regr_r2	float8_regr_slope
float8_regr_sxx	float8_regr_sxy	float8_regr_syy	float8_stddev_pop	float8_stddev_samp	float8_var_pop	float8_var_samp
float84div	float84eq	float84ge	float84gt	float84le	float84lt	float84mi
float84mul	float84ne	float84pl	float84abs	float84div	float84eq	float84ge
float8gt	float8in	float8larger	float8le	float8lt	float8mi	float8mul
float8ne	float8out	float8pl	float8recv	float8send	float8smaller	float8um
float8up	floor	flt4_mul_cash	flt8_mul_cash	fmgr_validator	fmgr_internal_validator	fmgr_sql_validator
format	format_type	gb18030_to_utf8	gbk_to_utf8	generate_series	generate_subscripts	get_bit
get_byte	get_current_ts_config	get_global_asp	get_largest_table_name	-	-	-
gtsquery_compress	gtsquery_consistent	gtsquery_decompress	gtsquery_penalty	gtsquery_picksplit	gtsquery_same	gtsquery_union
gtsvector_compress	gtsvector_consistent	gtsvector_decompress	gtsvector_penalty	gtsvector_picksplit	gtsvector_same	gtsvector_union
gtsvectorin	gtsvectorout	has_table_privilege	has_type_privilege	hash_aclitem	hashbeginscan	hashbuild

hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap	hashgettupl	hashinsert
hashint2vector	hashint4	hashint8	hashmacaddr	hashmarkpos	hashname	hashoid
hashoidvector	hashoptions	hashrescan	hashrestrpos	hashtext	hashvacuumcleanu	hashvarlena
host	hostmask	iclikejoinsel	iclike	iclikejoinsel	iclike	icregexeqjoinsel
icregexeqsel	icregexnejoinsel	icregexne	inet_client_addr	inet_client_port	inet_in	inet_out
inet_recv	inet_send	inet_server_addr	inet_server_port	inetand	inetmi	inetmi_int8
inetnot	inetor	inetpl	initcap	int2_accum	int2_avg_accum	int2_mul_cash
int2_sum	int24div	int24eq	int24ge	int24gt	int24le	int24lt
int24mi	int24mul	int24ne	int24pl	int28div	int28eq	int28ge
int28gt	int28le	int28lt	int28mi	int28mul	int28ne	int28pl
int2abs	int2and	int2div	int2eq	int2ge	int2gt	int2in
int2larger	int2le	int2lt	int2mi	int2mod	int2mul	int2ne
int2not	int2or	int2out	int2pl	int2recv	int2send	int2shl
int2shr	int2smaller	int2um	int2up	int2vector	int2vectorin	int2vectorout
int2vectorrecv	int2vectorsend	int2xor	int4_accum	int4_avg_accum	int4_mul_cash	int4_sum
int42div	int42eq	int42ge	int42gt	int42le	int42lt	int42mi
int42mul	int42ne	int42pl	int48div	int48eq	int48ge	int48gt

int48le	int48lt	int48mi	int48mul	int48ne	int48pl	int4abs
int4and	int4div	int4eq	int4ge	int4gt	int4in	int4inc
int4larger	int4le	int4lt	int4mi	int4mod	int4mul	int4ne
int4not	int4or	int4out	int4pl	int4range	int4range_canonical	int4range_subdiff
int4recv	int4send	int4shl	int4shr	int4smaller	int4um	int4up
int4xor	int8	int8_avg	int8_avg_accum	int8_avg_collect	int8_mul_cash	int8_sum
int8_sum_to_int8	int8+1635:1668_accum	int82div	int82eq	int82ge	int82gt	int82le
int82lt	int82mi	int82mul	int82ne	int82pl	int84div	int84eq
int84ge	int84gt	int84le	int84lt	int84mi	int84mul	int84ne
int84pl	int8abs	int8and	int8div	int8eq	int8ge	int8gt
int8in	int8inc	int8inc_any	int8inc_float8_float8	int8larger	int8le	int8lt
int8mi	int8mod	int8mul	int8ne	int8not	int8or	int8out
int8pl	int8pl_inet	int8range	int8range_canonical	int8range_subdiff	int8recv	int8send
int8shl	int8shr	int8smaller	int8sum	int8up	int8xor	integer_pl_date
inter_lb	inter_sb	inter_sl	internal_in	internal_out	interval	interval_accum
interval_avg	interval_cmp	interval_collect	interval_div	interval_eq	interval_ge	interval_gt
interval_has	interval_in	interval_larger	interval_le	interval_lt	interval_mi	interval_mul

interval_ne	interval_out	interval_pl	interval_pl_date	interval_pl_time	interval_pl_timestamp	interval_pl_timestampz
interval_pl_timestz	interval_recv	interval_send	interval_smaller	interval_transform	interval_um	intervaltypmodin
intervaltypmodout	intinterval	isexists	ishorizontal	iso_to_koi8r	iso_to_mic	iso_to_win1251
iso_to_win866	iso8859_1_to_utf8	iso8859_to_utf8	isparallel	isperp	isvertical	johab_to_utf8
jsonb_in	jsonb_out	jsonb_recv	jsonb_send	-	-	-
json_in	json_out	json_recv	json_send	justify_days	justify_hours	justify_interval
koi8r_to_iso	koi8r_to_mic	koi8r_to_utf8	koi8r_to_win1251	koi8r_to_win866	koi8u_to_utf8	language_handler_in
language_handler_out	latin1_to_mic	latin2_to_mic	latin2_to_win1250	latin3_to_mic	latin4_to_mic	like_escape
likejoinsel	likesel	line	line_distance	line_eq	line_horizontal	line_in
line_interpt	line_intersect	line_out	line_parallel	line_perp	line_recv	line_send
line_vertical	ln	lo_close	lo_create	lo_create	lo_export	lo_import
lo_lseek	lo_open	lo_tell	lo_truncate	lo_unlink	log	loread
lower	lower_inc	lower_inf	lowrite	lpad	lseg	lseg_center
lseg_distance	lseg_eq	lseg_ge	lseg_gt	lseg_horizontal	lseg_in	lseg_interpt
lseg_intersect	lseg_le	lseg_length	lseg_lt	lseg_ne	lseg_out	lseg_parallel
lseg_perp	lseg_recv	lseg_send	lseg_vertical	ltrim	macaddr_and	macaddr_cmp
macaddr_eq	macaddr_ge	macaddr_gt	macaddr_in	macaddr_le	macaddr_lt	macaddr_ne

macaddr_no t	macaddr _or	macaddr_ out	macad dr_recv	macadd r_send	makea clitem	masklen
max	md5 The MD5 encryptio n algorith m has lower security and poses security risks. Therefor e, you are advised to use a more secure encryptio n algorith m.	mic_to_bi g5	mic_to _euc_c n	mic_to_ euc_jp	mic_to _euc_k r	mic_to_euc_ tw
mic_to_iso	mic_to_k oi8r	mic_to_la tin1	mic_to _latin2	mic_to_l atin3	mic_to _latin4	mic_to_sjis
mic_to_win1 250	mic_to_w in1251	mic_to_w in866	min	mktinte rval	money	mul_d_inter val
name	nameeq	namege	nameg t	nameicli ke	namei cnlike	nameicrege xeq
nameicrege xne	namein	namele	nameli ke	namelt	namen e	namenlike
nameout	namerec v	namereg exeq	namer egexne	namese nd	neqjoi nsel	neqsel
network_cm p	network_ eq	network_ ge	networ k_gt	network _le	netwo rk_lt	network_ne
network_su b	network_ subeq	network_ sup	networ k_supe q	nlikejoin sel	nlikese l	numeric
numeric_ab s	numeric_ accum	numeric_ add	numeri c_avg	numeric _avg_ac cum	numeri c_avg_ collec t	numeric_cm p

numeric_collect	numeric_div	numeric_div_trunc	numeric_eq	numeric_exp	numeric_fac	numeric_ge
numeric_gt	numeric_in	numeric_inc	numeric_larger	numeric_le	numeric_ln	numeric_log
numeric_lt	numeric_mod	numeric_mul	numeric_ne	numeric_out	numeric_power	numeric_recv
numeric_send	numeric_smaller	numeric_sortsupport	numeric_sqrt	numeric_stddev_pop	numeric_stddev_sample	numeric_sub
numeric_transform	numeric_uminus	numeric_uplus	numeric_var_pop	numeric_var_sample	numericypmodin	numericypmodout
numrange_subdiff	oid	oideq	oidge	oidgt	oidin	oidlarger
oidle	oidlt	oidne	oidout	oidrecv	oidsend	oidsmaller
oidvectoreq	oidvectorge	oidvectorgt	oidvectorin	oidvectorle	oidvectorlt	oidvectorne
oidvectorout	oidvectorrecv	oidvectorsend	oidvectorotypes	on_pb	on_pl	on_ppath
on_ps	on_sb	on_sl	opaque_in	opaque_out	ordered_set_transition	overlaps
overlay	path	path_add	path_add_pt	path_center	path_contain_pt	path_distance
path_div_pt	path_in	path_inter	path_length	path_mul_pt	path_neq	path_nge
path_n_gt	path_n_le	path_n_lt	path_n_points	path_out	path_recv	path_send
path_sub_pt	percentile_cont	percentile_cont_float8_final	percentile_cont_interval_final	pg_char_to_encoding	pg_cursor	pg_encoding_max_length
pg_encoding_to_char	pg_extension_config_dump	-	-	pg_node_tree_in	pg_node_tree_out	pg_node_tree_recv

pg_node_tree_send	pg_prepared_statement	pg_prepared_xact	pg_notify	pg_stat_get_wal_receiver	pg_show_all_settings	pg_stat_get_bgwriter_stat_reset_time
pg_stat_get_buf_fsync_backend	pg_stat_get_checkpoint_sync_time	pg_stat_get_checkpoint_write_time	pg_stat_get_dblink_read_time	pg_stat_get_db_blk_write_time	pg_stat_get_db_conflict_all	pg_stat_get_db_conflict_bufferpin
pg_stat_get_db_conflict_snapshot	pg_stat_get_db_conflict_startup_deadlock	pg_switch_xlog	xpath	pg_timezone_abbrs	pg_timezone_names	pgxc_node_str
plpgsql_call_handler	plpgsql_inline_handler	plpgsql_validator	point_above	point_add	point_below	point_distance
point_div	point_eq	point_horiz	point_in	point_left	point_mul	point_ne
point_out	point_rect	point_right	point_s_end	point_sub	point_vert	poly_above
poly_below	poly_center	poly_contain	poly_contain_pt	poly_contained	poly_distance	poly_in
poly_left	poly_npoints	poly_out	poly_overabove	poly_overbelow	poly_overlap	poly_overleft
poly_overright	poly_rectv	poly_right	poly_same	poly_solid	polygon	position
positionjoin	positions	postgres_fdw_validator	pow	power	prsd_end	prsd_headline
prsd_lextype	prsd_nexttoken	prsd_start	pt_contained_circle	pt_contained_poly	query_to_xml	query_to_xml_and_xml_schema
query_to_xmlschema	quote_ident	quote_literal	quote_nullable	radians	radius	random
range_adjacent	range_after	range_before	range_cmp	range_contained_by	range_contains	range_contains_elem
range_eq	range_ge	range_gt	range_in	range_intersect	range_le	range_lt

range_minus	range_ne	range_ou	range_	range_o	range_	range_recv
	panalyze	t	overlap	verleft	overrig	
			s		ht	
range_send	range_ty	range_uni	rank	record_e	record_	record_gt
	panalyze	on		q	_ge	
record_in	record_le	record_lt	record_	record_	record_	record_send
			ne	out	_recv	
regclass	regclassi	regclasso	regclas	regclass	regconf	regconfigou
	n	ut	srecv	send	igin	t
regconfigrec	regconfig	regdictio	regdicti	regdicti	regdicti	regexeqjoins
v	send	naryin	onaryo	onaryre	onary	el
			ut	cv	send	
regexeqsel	regexnej	regexnes	regexp_	regexp_	regexp_	regexp_split
	oinsel	el	_match	_replace	_split_	_to_table
			es		_to_arr	
					ay	
regoperatori	regopera	regoperat	regope	regoperi	regope	regoperrecv
n	torout	torrecv	operatorse	n	rou	
			nd			
regopersend	regproce	regproce	regpro	regproc	regpro	regprocout
	durein	dureout	cedurer	edurese	cin	
			ecv	nd		
regprocrecv	regprocs	regr_avgx	regr_av	regr_co	regr_in	regr_r2
	end		gy	unt	tercept	
regr_slope	regr_sxx	regr_sxy	regr_sy	regtypei	regtyp	regtyperecv
			y	n	eout	
regtypesend	reltime	reltimeeq	reltime	reltimeg	reltim	reltimele
			ge	t	ein	
reltimelt	reltimen	reltimeou	reltime	reltimes	repeat	replace
	e	t	recv	end		
reverse	RI_FKey_	RI_FKey_c	RI_FKe	RI_FKey_	RI_FKe	RI_FKey_noa
	cascade_	ascade_u	y_chec	_check_	y_noac	ction_upd
	del	pd	k_ins	upd	tion_d	
					el	
RI_FKey_restr	RI_FKey_	RI_FKey_s	RI_FKe	RI_FKey_	RI_FKe	right
ict_del	restrict_u	etdefault	y_setde	_setnull	y_setn	
	pd	_del	fault_u	_del	ull_up	
			pd		d	
round	row_num	row_to_js	rpadd	rtrim	scalar	scalargtsel
	ber	on			gtjoins	
					el	

scalarltojoinse	scalarltsel	schema_to_xml	schema_to_xml_and_xmlschema	schema_to_xmlschema	session_user	set_bit
set_byte	set_config	set_masklen	shift_jis_2004_to_euc_jis_2004	shift_jis_2004_to_utf8	sjis_to_euc_jp	sjis_to_mic
sjis_to_utf8	smgrin	smgrout	spg_kd_choose	spg_kd_config	spg_kd_inner_consistent	spg_kd_picksplit
spg_quad_choose	spg_quad_config	spg_quad_inner_consistent	spg_quad_leaf_consistent	spg_quad_picksplit	spg_text_choose	spg_text_config
spg_text_inner_consistent	spg_text_leaf_consistent	spg_text_picksplit	spgbeginscan	spgbuild	spgbuilderempty	spgbulkdelete
spgcanreturn	spgcostestimate	spgendscan	spggetbitmap	spggettuple	spginsert	spgmarkpos
spgoptions	spgrescan	spgrestrpos	spgvacuumcleanup	stddev	stddev_pop	stddev_sample
string_agg	string_agg_finalfn	string_agg_transfn	strip	sum	suppress_redundant_updates_trigger	table_to_xml
table_to_xml_and_xmlschema	table_to_xmlschema	tan	text	text_ge	text_gt	text_larger
text_le	text_lt	text_pattern_ge	text_pattern_gt	text_pattern_le	text_pattern_lt	text_smaller
textanycat	textcat	texteq	texticlike	texticlike	texticregexeq	texticregexne
textin	textlike	textne	textnllike	textout	textrecv	textregexeq

textregexne	textsend	thesaurus_init	thesaurus_lexize	tideq	tidge	tidgt
tidin	tidlarger	tidle	tidlt	tidne	tidout	tidrecv
tidsend	tidsmaller	time	time_cmp	time_eq	time_ge	time_gt
time_hash	time_in	time_larger	time_le	time_lt	time_mi_interval	time_mi_time
time_ne	time_out	time_pl_interval	time_recv	time_send	time_smaller	time_transform
timedate_pl	timemi	timepl	timestamp	timestamp_cmp	timestamp_date	timestamp_cmp_timestampz
timestamp_eq	timestamp_eq_date	timestamp_eq_timestampz	timestamp_ge	timestamp_ge_date	timestamp_ge_timestampz	timestamp_gt
timestamp_gt_date	timestamp_gt_timestampz	timestamp_hash	timestamp_in	timestamp_larger	timestamp_le	timestamp_le_date
timestamp_le_timestampz	timestamp_lt	timestamp_lt_date	timestamp_lt_timestampz	timestamp_mi	timestamp_mi_interval	timestamp_ne
timestamp_ne_date	timestamp_ne_timestampz	timestamp_out	timestamp_pl_interval	timestamp_recv	timestamp_send	timestamp_smaller
timestamp_sortsupport	timestamp_transform	timestamp_typmodin	timestamp_typmodout	timestamp_tz	timestamp_tz_cmp	timestamp_tz_cmp_date
timestampz_cmp_timestamp	timestampz_eq	timestampz_eq_date	timestampz_eq_timestamp	timestampz_ge	timestampz_ge_date	timestampz_ge_timestamp
timestampz_gt	timestampz_gt_date	timestampz_gt_timestamp	timestampz_in	timestampz_larger	timestampz_le	timestampz_le_date

timestampz_le_timestamp	timestampz_lt	timestampz_lt_date	timestampz_lt_timestamp	timestampz_mi	timestampz_mi_interval	timestampz_ne
timestampz_ne_date	timestampz_ne_timestamp	timestampz_out	timestampz_pl_interval	timestampz_recv	timestampz_send	timestampz_smaller
timestampztypmodin	timestampztypmodout	timetypmodin	timetypmodout	timetz	timetz_cmp	timetz_eq
timetz_ge	timetz_gt	timetz_hash	timetz_in	timetz_larger	timetz_le	timetz_lt
timetz_mi_interval	timetz_ne	timetz_out	timetz_pl_interval	timetz_recv	timetz_send	timetz_smaller
timetzdate_pl	timetztypmodin	timetztypmodout	timezone(2069)	timezone(1159)	timezone(2037)	timezone(2070)
timezone(1026)	timezone(2038)	tintervalc_t	tintervalc_eq	tintervalc_ge	tintervalc_gt	tintervalc_in
tintervalle	tintervalle_eq	tintervalleenge	tintervallelength	tintervallelenle	tintervallelength	tintervallen
tintervallt	tintervalle	tintervalleout	tintervalleov	tintervallerecv	tintervalle	tintervalle
tintervalstart	to_ascii(1845)	to_ascii(1847)	to_ascii(1846)	trigger_in	trigger_out	ts_match_qv
ts_match_tq	ts_match_tt	ts_match_vq	ts_rank	ts_rank_cd	ts_rewrite	ts_stat
ts_token_type	ts_tyanalyze	tsmatchjoin	tsmatchsel	tsq_mcontained	tsq_mcontains	tsquery_and
tsquery_cmp	tsquery_eq	tsquery_ge	tsquery_gt	tsquery_le	tsquery_lt	tsquery_ne
tsquery_not	tsquery_or	tsqueryin	tsqueryout	tsqueryrecv	tsquerysend	tsrange
tsrange_subdiff	tstzrange	tstzrange_subdiff	tsvector_cmp	tsvector_concat	tsvector_eq	tsvector_ge

tsvector_gt	tsvector_le	tsvector_lt	tsvector_ne	tsvector_update_trigger	tsvector_update_trigger_column	tsvectorin
tsvectorout	tsvectorrecv	tsvectorsend	txid_current	txid_current_snapshot	txid_snapshot_in	txid_snapshot_out
txid_snapshot_recv	txid_snapshot_send	txid_snapshot_xip	txid_snapshot_xmax	txid_snapshotxmin	txid_visible_in_snapshot	uhc_to_utf8
unique_key_recheck	unknownin	unknownout	unknownrecv	unknownsend	-	utf8_to_big5
utf8_to_euc_cn	utf8_to_euc_jis_2004	utf8_to_euc_jp	utf8_to_euc_kr	utf8_to_euc_tw	utf8_to_gb18030	utf8_to_gbk
utf8_to_iso8859	utf8_to_iso8859_1	utf8_to_johab	utf8_to_koi8r	utf8_to_koi8u	utf8_to_shift_jis_2004	utf8_to_sjis
utf8_to_uhc	utf8_to_win	uuid_cmp	uuid_eq	uuid_ge	uuid_gt	uuid_hash
uuid_in	uuid_le	uuid_lt	uuid_ne	uuid_out	uuid_recv	uuid_send
var_pop	var_sample	varbit	varbit_in	varbit_out	varbit_recv	varbit_send
varbit_transform	varbitcmp	varbiteq	varbitge	varbitgt	varbitle	varbitlt
varbitne	varbittypmodin	varbittypmodout	varchar	varchar_transform	varcharin	varcharout
varcharrecv	varcharsend	varchartypmodin	varchartypmodout	variance	void_in	void_out
void_recv	void_send	win_to_utf8	win1250_to_latin2	win1250_to_mic	win1251_to_iso	win1251_to_koi8r
win1251_to_mic	win1251_to_win866	win866_to_iso	win866_to_koi8r	win866_to_mic	win866_to_win1251	xideq

xideqint4	xidin	xidout	xidrecv	xidsend	xml	xml_in
xml_is_well_formed	xml_is_well_formed_content	xml_is_well_formed_document	xml_out	xml_recv	xml_send	xmlagg
xmlcomment	xmlconcat2	xmlexists	xmlvalidate	-	-	-

 **NOTE**

To prevent memory bloat, it is recommended that the number of nested layers of the concat, concat_ws, textanycat, anytextcat and format functions be less than or equal to five.

Internal Functions

The following table lists the functions used by GaussDB to implement internal system functions. You are advised not to use these functions. If you need to use them, contact Huawei technical support.

- **spread_collect**
Description: Calculates the difference between the maximum and minimum values in a certain period of time. This function is used for data collection of aggregate functions.
Parameters: s real[], v real[]
Return type: real[]
- **spread_final**
Description: Calculates the difference between the maximum and minimum values in a certain period of time. This function is used for the final data processing of the aggregate function.
Parameter: s real[]
Return type: real
- **spread_internal**
Description: Calculates the difference between the maximum and minimum values in a certain period of time. This function is used for the process of aggregate functions.
Parameters: s real[], v real
Return type: real[]
- **xidin4**
Description: Inputs a 4-byte xid.
Parameter: cstring
Return type: xid32
- **set_hashbucket_info**
Description: Sets hash bucket information.
Parameter: text

- Return type: Boolean
- `gap_fill_internal`
Description: Returns the first non-NULL value in the parameter list.
Parameter: `s anyelement, v anyelement`
Return type: `anyelement`
 - `int1send`
Description: Packs unsigned 1-byte integers into the internal data buffer stream.
Parameter: `tinyint`
Return type: `bytea`
 - `is_contain_namespace`
Description: Searches for the table name and namespace split location. If no namespace exists, 0 is returned.
Parameter: `relationname name`
Return type: `integer`
 - `is_oid_in_group_members`
Description: Not supported
Parameter: `node_oid oid, group_members oidvector_extend`
Return type: `Boolean`
 - `isubmit_on_nodes_internal`
Description: Not supported
Parameter: `job bigint, node_name name, database name, what text, next_date timestamp without time zone, job_interval text`
Return type: `integer`
 - `listagg`
Description: Specifies aggregate functions of the list type.
Parameters: `smallint, text`
Return type: `text`
 - `log_fdw_validator`
Description: Specifies validation functions.
Parameters: `text[], oid`
Return type: `void`
 - `nvarchar2typmodin`
Description: Obtains the typmod information of the varchar type.
Parameter: `cstring[]`
Return type: `integer`
 - `nvarchar2typmodout`
Description: Obtains the typmod information of the varchar type, constructs a character string, and returns the character string.
Parameter: `integer`
Return type: `cstring`

- `pg_nodes_memmon`
Description: Not supported
Parameter: nan
Return type: innernname text, innerusedmem bigint, innertopctxt bigint, nname text, usedmem text, sharedbuffercache text, topcontext text
- `read_disable_conn_file`
Description: Reads forbidden connection files.
Parameter: nan
Return type: disconn_mode text, disconn_host text, disconn_port text, local_host text, local_port text, redo_finished text
- `regex_like_m`
Description: Specifies the regular expression match, which is used to determine whether a character string complies with a specified regular expression.
Parameters: text, text
Return type: Boolean
- `update_pgjob`
Description: Updates a job.
Parameters: bigint, "char", bigint, timestamp without time zone, smallint, text
Return type: void
- `enum_cmp`
Description: Specifies the enumeration comparison function, which is used to determine whether two enumeration classes are equal and determine their relative sizes.
Parameters: anyenum, anyenum
Return type: integer
- `enum_eq`
Description: Specifies the enumeration comparison function, which is used to implement the equal sign (=).
Parameters: anyenum, anyenum
Return type: Boolean
- `enum_first`
Description: Returns the first element in the enumeration class.
Parameter: anyenum
Return type: anyenum
- `enum_ge`
Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>) and equal sign (=).
Parameters: anyenum, anyenum
Return type: Boolean
- `enum_gt`

Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>).

Parameters: anyenum, anyenum

Return type: Boolean

- enum_in

Description: Specifies the enumeration comparison function, which is used to determine whether an element is in an enumeration class.

Parameters:cstring, OID

Return type: anyenum

- enum_larger

Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>).

Parameters: anyenum, anyenum

Return type: anyenum

- enum_last

Description: Returns the last element in the enumeration class.

Parameter: anyenum

Return type: anyenum

- enum_le

Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<) and equal sign (=).

Parameters: anyenum, anyenum

Return type: Boolean

- enum_lt

Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<).

Parameters: anyenum, anyenum

Return type: Boolean

- enum_smaller

Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<).

Parameters: anyenum, anyenum

Return type: Boolean

- node_oid_name

Description: Not supported

Parameter: oid

Return type:cstring

- pg_buffercache_pages

Description: Reads data from the shared buffer.

Parameter: nan

Return type:bufferid integer, relfilenode oid, bucketid smallint, storage_type oid, reltablespace oid, reldatabase oid, relforknumber smallint, relblocknumber bigint, isdirty Boolean, usage_count smallint

- `pg_check_xidlimit`
Description: Checks whether nextxid is greater than or equal to xidwarnlimit.
Parameter: nan
Return type: Boolean
- `pg_comm_delay`
Description: Displays the delay status of the communications library of a single DN.
Parameter: nan
Return type: text, text, integer, integer, integer, integer
- `pg_comm_rcv_stream`
Description: Displays the receiving stream status of all communication libraries on a single DN.
Parameter: nan
Return type: text, bigint, text, bigint, integer, integer, integer, text, bigint, integer, integer, integer, bigint, bigint, bigint, bigint, bigint
- `pg_comm_send_stream`
Description: Displays the sending stream status of all communication libraries on a single DN.
Parameter: nan
Return type: text, bigint, text, bigint, integer, integer, integer, text, bigint, integer, integer, integer, bigint, bigint, bigint, bigint, bigint
- `pg_comm_status`
Description: Displays the communication status of a single DN.
Parameter: nan
Return type: text, integer, integer, bigint, bigint, bigint, bigint, bigint, integer, integer, integer, integer, integer
- `pg_log_comm_status`
Description: Prints some logs on the DN.
Parameter: nan
Return type: Boolean
- `pg_parse_clog`
Description: Parses clog to obtain the status of xid.
Parameter: nan
Return type: xid xid, status text
- `pg_pool_ping`
Description: Sets PoolerPing.
Parameter: Boolean
Return type: SETOF Boolean
- `pg_pool_validate`
Description: Compares fields in the **pgxc_node** system catalog to check whether a connection is available.
Parameter: clear Boolean, co_node_name cstring

Return type: pid bigint, node_name text

- pg_resume_bkp_flag

Description: Obtains the delay Xlog flag for backup and restoration.

Parameter: slot_name name

Return type: start_backup_flag Boolean, to_delay Boolean, ddl_delay_recycle_ptr text, rewind_time text

- pg_stat_get_pooler_status

Description: Queries the cache connection status in the pooler.

Parameter: nan

Return type: text, text, bigint, text, bigint, Boolean, text, bigint, bigint, bigint, bigint, bigint

Table 7-137 PG_STAT_GET_POOLER_STATUS columns

Name	Type	Description
database_name	OUT text	Database name.
user_name	OUT text	Username.
tid	OUT bigint	In non-thread pool logic, this parameter indicates the ID of the thread connected to the CN. In thread pool logic, this parameter indicates the ID of the session connected to the CN.
pgoptions	OUT text	Database connection option. For details, see the options column in Connection Parameters .
node_oid	OUT bigint	OID of the node connected.
in_use	OUT boolean	Specifies whether the connection is currently used. <ul style="list-style-type: none"> • t (true): The connection is in use. • f (false): The connection is not in use.
session_params	OUT text	GUC session parameter delivered by the connection.
fdsock	OUT bigint	Local socket.
remote_pid	OUT bigint	Peer thread ID
used_count	OUT bigint	Number of reuse times of a connection.
idx	OUT bigint	Peer DN ID in the local CN.

Name	Type	Description
streamid	OUT bigint	Stream ID in the physical connection.

- gs_validate_ext_listen_ip

Description: Connects to DNs, queries invalid service threads on DNs connected to the original extended IP address, and clears the threads.

Parameter: For details, see [Table 7-138](#).

Return value: bigint pid, text node_name

Note: This function is used only when old connections still exist on the original extended IP address after the extended IP address is reloaded. This function clears the threads where the original IP connections are located. Executing this function will clear communication listening on the extended IP address. Exercise caution when performing O&M operations. Currently, this command can be executed only by connecting to DNs and delivered by the administrator.

Table 7-138 GS_VALIDATE_EXT_LISTEN_IP columns

Name	Type	Description
clear	IN cstring	Specifies whether to clear data. The options are as follows: on : clear; off : not clear.
validate_node_name	IN cstring	Name of the DN where the extended IP address connections to be cleared are located.
validate_ip	IN cstring	Extended IP address whose connections are to be cleared.
pid	OUT bigint	ID of the service thread where the extended IP address connections to be cleared are located.
node_name	OUT text	Name of the instance to which the extended IP address connections to be cleared belong.

- gs_comm_listen_address_ext_info

Description: Displays the DFX information about the extended IP address configured for **listen_address_ext** connected to the current instance.

Parameter: nan

Return type: text node_name, text app, bigint tid, integer lwtid, bigint query_id, integer socket, text remote_ip, text remote_port, text local_ip, text local_port

Table 7-139 GS_COMM_LISTEN_ADDRESS_EXT_INFO columns

Name	Type	Description
node_name	OUT text	Name of the current instance.
app	OUT text	Client connected to the DN.
tid	OUT bigint	Thread ID of the current thread.
lwtid	OUT integer	Lightweight thread ID of the current thread.
query_id	OUT bigint	Query ID of the current thread.
socket	OUT integer	Socket FD of the current physical connection.
remote_ip	OUT text	Peer IP address of the current connection.
remote_port	OUT text	Peer port of the current connection.
local_ip	OUT text	Local IP address of the current connection.
local_port	OUT text	Local port of the current connection.

- `gs_get_global_listen_address_ext_info()`
Description: Queries the global extended IP address configuration on the CN.
Parameter: For details, see [Table 7-140](#).
Return type: text node_name, text host, text port, text ext_listen_ip
Note: If the input parameter of the function is **all**, the function depends on the **ext_ip_info** static configuration file. If the file is abnormal, use the CM/OM tool to rectify the fault. In this case, the configuration cannot be updated in real time. You can set the input parameter to **primary** to obtain the configuration information about all primary DNs.

Table 7-140 GS_GET_GLOBAL_LISTEN_ADDRESS_EXT_INFO columns

Name	Type	Description
dn_mode	IN cstring	Specifies the range of DNs to be displayed. If this parameter is set to null , all DNs are queried by default.
node_name	OUT text	DN name.
host	OUT text	Listening IP address of the DN.
port	OUT text	Listening port of the DN.
ext_listen_ip	OUT text	Extended listening IP address of the DN.

- `gs_get_listen_address_ext_info()`
Description: Queries the extended IP address configuration of the current instance.
Parameter: nan
Return type: text node_name, text host, bigint port, text ext_listen_ip

Table 7-141 GS_GET_LISTEN_ADDRESS_EXT_INFO

Name	Type	Description
node_name	OUT text	DN name.
host	OUT text	Listening IP address of the DN.
port	OUT bigint	Listening port of the DN.
ext_listen_ip	OUT text	Extended listening IP address of the DN.

- `psortoptions`
Description: Returns the psort attribute.
Parameters: text[], Boolean
Return type: bytea
- `remove_job_class_depend`
Description: Removes the job dependency.
Parameter: oid
Return type: void
- `xideq4`
Description: Compares two xids to check whether they are the same.
Parameters: xid32, xid32
Return type: Boolean
- `xideqint8`
Description: Compares two xids to check whether they are the same.
Parameters: xid, bigint
Return type: Boolean
- `xidlt`
Description: Returns whether `xid1 < xid2` is true.
Parameters: xid, xid
Return type: Boolean
- `xidlt4`
Description: Returns whether `xid1 < xid2` is true.
Parameters: xid32, xid32
Return type: Boolean
- `get_local_cont_query_stat`

Description: Obtains the statistics of a specified continuous computing view on the local node.

Parameter: cq_id oid

Return value type: cq oid, w_in_rows int8, w_in_bytes int8, w_out_rows int8, w_out_bytes int8, w_pendings int8, w_errors int8, r_in_rows int8, r_in_bytes int8, r_out_rows int8, r_out_bytes int8, r_errors int8, c_in_rows int8, c_in_bytes int8, c_out_rows int8, c_out_bytes int8, c_pendings int8, c_errors int8

- `get_local_cont_query_stats`

Description: Obtains all continuous computing view statistics of the local node.

Parameter: nan

Return value type: cq oid, w_in_rows int8, w_in_bytes int8, w_out_rows int8, w_out_bytes int8, w_pendings int8, w_errors int8, r_in_rows int8, r_in_bytes int8, r_out_rows int8, r_out_bytes int8, r_errors int8, c_in_rows int8, c_in_bytes int8, c_out_rows int8, c_out_bytes int8, c_pendings int8, c_errors int8

- `get_cont_query_stats`

Description: Obtains statistics about all continuous computing views on each DN.

Parameter: nan

Return value type: node name, cq oid, w_in_rows int8, w_in_bytes int8, w_out_rows int8, w_out_bytes int8, w_pendings int8, w_errors int8, r_in_rows int8, r_in_bytes int8, r_out_rows int8, r_out_bytes int8, r_errors int8, c_in_rows int8, c_in_bytes int8, c_out_rows int8, c_out_bytes int8, c_pendings int8, c_errors int8

- `reset_local_cont_query_stat`

Description: Resets the statistics of a specified continuous computation view on the local node.

Parameter: cq_id oid

Return type: Boolean

- `reset_local_cont_query_stats`

Description: Resets association statistics on the specified continuous computation view of the local node.

Parameter: cq_id oid

Return type: Boolean

- `reset_cont_query_stats`

Description: Resets the continuous computation view statistics corresponding to the STREAM object on each DN.

Parameter: stream_id oid

Return type: Boolean

- `check_cont_query_schema_changed`

Description: Determines the schema change status of a specified continuous computation view.

Parameter: cq_id oid

Return type: Boolean

- gs_get_standby_cluster_barrier_status**

Description: Queries the barrier log replay information of the standby CN or DN, including the latest received barrier, LSN of the latest received barrier, barrier played back last time, and target barrier to be played back.

Parameter: nan

Return type: barrier_id text, barrier_lsn text, recovery_id text, target_id text

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operate_mode**.
- gs_set_standby_cluster_target_barrier_id**

Description: Set the target barrier to be replayed.

Parameter: barrier_id

Return type: target_id text

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operate_mode**.
- gs_query_standby_cluster_barrier_id_exist**

Description: Queries whether the specified barrier is received by the standby node.

Parameter: barrier_id

Return type: Boolean

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operate_mode**.
- standby_read_status**

Description: Obtains the CSN-based snapshot information of all DNs in a cluster, the difference between the primary and standby DNs, and the CSN-based snapshot information used for the last read query on the standby node.

Parameter: nan

Return type: text, text, text, integer, bigint, bigint, bigint

Table 7-142 standby_read_status columns

Name	Type	Description
name	text	DN name
type	text	DN type (primary for the primary node and standby for the standby node)
host	text	IP address used by the DN
port	integer	Port used by the DN
collected_csn	bigint	CSN-based snapshot information of the DN (For the primary node, the value is 0 . For the standby node, the value is the collected CSN.)
delay	bigint	Difference between the primary and standby DNs. The unit is ms.

Name	Type	Description
visited_csn	bigint	CSN information used for the last read query on the standby node. (This parameter is valid only for the standby node where the last query is executed.)

- `gs_shutdown_cross_region_walsenders`
Description: Interrupts cross-cluster streaming replication.
Parameter: nan
Return type: void
Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operate_mode**.
The following stream functions exist but are not supported. You are advised not to use them.
`streaming_int8_avg_gather`, `streaming_numeric_avg_gather`,
`streaming_float8_avg_gather`, `streaming_interval_avg_gather`,
`streaming_int8_sum_gather`, and `streaming_int2_int4_sum_gather`
- `is_dblink_in_transaction`
Description: Checks whether a database link corresponding to an OID is used in the current transaction.
Parameter: oid
Return type: Boolean
- `dblink_has_updatasent`
Description: Checks whether DML statements are sent using a database link corresponding to an OID in the current transaction and are not committed.
Parameter: oid
Return type: Boolean
- `get_last_xmin_by_oid`
Description: Obtains the maximum **xmin** value of all columns in a table based on the table OID.
Parameter: oid
Return type: xid
- `get_relid_by_relname`
Description: Obtains the OID of a table based on the table name and renamespace.
Parameter: cstring, oid
Return type: oid

7.5.42 Obsolete Functions

The following functions in GaussDB have been discarded in the latest version:

- `gs_wlm_get_session_info`
- `gs_wlm_get_user_session_info`

- check_engine_status
- encode_plan_node
- model_train_opt
- gs_stat_get_wlm_plan_operator_info
- track_model_train_opt
- array_extend
- dbperf.global_slow_query_info
- dbperf.global_slow_query_info_bytime
- dbperf.global_slow_query_history
- pg_reload_conf
- pg_rotate_logfile
- gs_stat_ustore
- pv_compute_pool_workload()
- pgxc_log_comm_status(void)

7.6 Expressions

7.6.1 Simple Expressions

Logical Expressions

[Logical Operators](#) lists the operators and calculation rules of logical expressions.

Comparative Expressions

[Comparison Operators](#) lists the common comparative operators.

In addition to comparative operators, you can also use the following sentence structure:

- BETWEEN operator
a BETWEEN x AND y is equivalent to **a >= x AND a <= y**.
a NOT BETWEEN x AND y is equivalent to **a < x OR a > y**.
- To check whether a value is null, use:
expression IS NULL
expression IS NOT NULL
or an equivalent (non-standard) sentence structure:
expression ISNULL
expression NOTNULL

NOTICE

- Do not write **expression=NULL** or **expression<>(=)NULL**, because **NULL** represents an unknown value, and these expressions cannot determine whether two unknown values are equal.
- Only the comparative expressions **IS NULL** and **IS NOT NULL** support data of XML type.

- is distinct from/is not distinct from
 - is distinct from
If the data types and values of A and B are different, the value is **true**.
If the data types and values of A and B are the same, the value is **false**.
Empty values are considered the same.
 - is not distinct from
If the data types and values of A and B are different, the value is **false**.
If the data types and values of A and B are the same, the value is **true**.
Empty values are considered the same.
- <=> NULL-safe equal operator
The comparison of NULL values is added on the basis of the comparison of '='. If neither the left nor right value of the operator is NULL, the result is the same as that of '='.
If the data types and values of A and B are different, the value is **false**.
If the data types and values of A and B are the same, the value is **true**.
Empty values are considered the same.

 **NOTE**

- The usage of the <=> operator is the same as that of **IS NOT DISTINCT FROM**.
- This operator is valid only when the database is compatible with the MySQL type (that is, **sql_compatibility** is set to '**MYSQL**'). Other types do not support this operator.

Examples

```
gaussdb=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
result
-----
t
(1 row)

gaussdb=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
result
-----
t
(1 row)

gaussdb=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
result
-----
f
(1 row)

gaussdb=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
result
```

```
-----  
f  
(1 row)  
  
gaussdb=# SELECT 2+2 IS NULL AS RESULT;  
result  
-----  
f  
(1 row)  
  
gaussdb=# SELECT 2+2 IS NOT NULL AS RESULT;  
result  
-----  
t  
(1 row)  
  
gaussdb=# SELECT 2+2 ISNULL AS RESULT;  
result  
-----  
f  
(1 row)  
  
gaussdb=# SELECT 2+2 NOTNULL AS RESULT;  
result  
-----  
t  
(1 row)  
  
gaussdb=# SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;  
result  
-----  
t  
(1 row)  
  
gaussdb=# SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;  
result  
-----  
f  
(1 row)  
  
gaussdb=# SELECT 1 <=> 1 AS RESULT;  
result  
-----  
t  
(1 row)  
  
gaussdb=# SELECT NULL <=> 1 AS RESULT;  
result  
-----  
f  
(1 row)  
  
gaussdb=# SELECT NULL <=> NULL AS RESULT;  
result  
-----  
t  
(1 row)
```

7.6.2 Condition Expressions

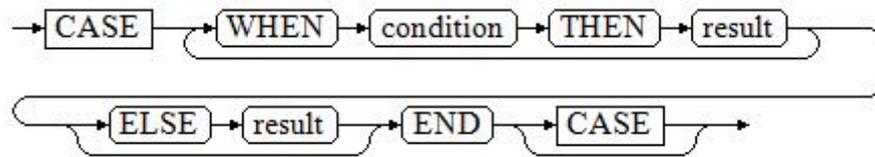
Data that meets the requirements specified by conditional expressions are filtered during SQL statement execution.

Conditional expressions include the following types:

- **CASE**
CASE expressions are similar to the **CASE** statements in other coding languages.

Figure 7-1 shows the syntax of a **CASE** expression.

Figure 7-1 case::=



A **CASE** clause can be used in a valid expression. **condition** is an expression that returns a value of Boolean type.

- If the result is **true**, the result of the **CASE** expression is the required result.
- If the result is false, the following **WHEN** or **ELSE** clauses are processed in the same way.
- If every **WHEN condition** is false, the result of the expression is the result of the **ELSE** clause. If the **ELSE** clause is omitted and has no match condition, the result is NULL.
- Operations on XML data are supported.

For example:

```

gaussdb=# CREATE TABLE case_when_t1(CW_COL1 INT) DISTRIBUTE BY HASH (CW_COL1);
gaussdb=# INSERT INTO case_when_t1 VALUES (1), (2), (3);
gaussdb=# SELECT * FROM case_when_t1;
a
---
1
2
3
(3 rows)

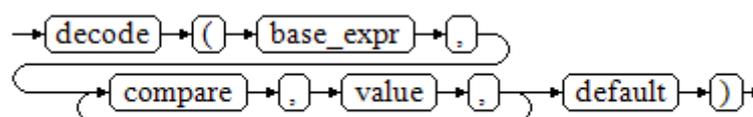
gaussdb=# SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN 'two'
ELSE 'other' END FROM case_when_t1 ORDER BY 1;
cw_col1 | case
-----+-----
1 | one
2 | two
3 | other
(3 rows)

gaussdb=# DROP TABLE case_when_t1;
  
```

- DECODE

Figure 7-2 shows the syntax of a **DECODE** expression.

Figure 7-2 decode::=



Compare each following **compare(n)** with **base_expr**, **value(n)** is returned if a **compare(n)** matches the **base_expr** expression. If **base_expr** does not match each **compare(n)**, the default value is returned.

Operations on XML data are supported.

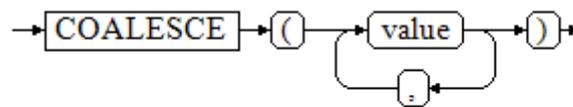
Conditional Expression Functions describes the examples.

```
gaussdb=# SELECT DECODE('A','A',1,'B',2,0);
case
-----
1
(1 row)
```

- COALESCE

Figure 7-3 shows the syntax of a **COALESCE** expression.

Figure 7-3 coalesce::=



COALESCE returns its first not-NULL value. If all the parameters are **NULL**, **COALESCE** will return **NULL**. This value is replaced by the default value when data is displayed. Like a **CASE** expression, **COALESCE** only evaluates the parameters that are needed to determine the result. That is, parameters to the right of the first not-**NULL** parameter are not evaluated.

Operations on XML data are supported.

Example:

```
gaussdb=# CREATE TABLE c_tabl(description varchar(10), short_description varchar(10), last_value
varchar(10))
DISTRIBUTE BY HASH (last_value);

gaussdb=# INSERT INTO c_tabl VALUES('abc', 'efg', '123');
gaussdb=# INSERT INTO c_tabl VALUES(NULL, 'efg', '123');

gaussdb=# INSERT INTO c_tabl VALUES(NULL, NULL, '123');

gaussdb=# SELECT description, short_description, last_value, COALESCE(description, short_description,
last_value) FROM c_tabl ORDER BY 1, 2, 3, 4;
description | short_description | last_value | coalesce
-----+-----+-----+-----
abc         | efg              | 123       | abc
           | efg              | 123       | efg
           |                  | 123       | 123
(3 rows)

gaussdb=# DROP TABLE c_tabl;
```

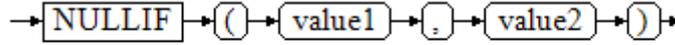
If **description** is not **NULL**, the value of **description** is returned. Otherwise, parameter **short_description** is calculated. If **short_description** is not **NULL**, the value of **short_description** is returned. Otherwise, parameter **last_value** is calculated. If **last_value** is not **NULL**, the value of **last_value** is returned. Otherwise, **none** is returned.

```
gaussdb=# SELECT COALESCE(NULL,'Hello World');
coalesce
-----
Hello World
(1 row)
```

- NULLIF

Figure 7-4 shows the syntax of a **NULLIF** expression.

Figure 7-4 nullif::=



Only if **value1** is equal to **value2** can **NULLIF** return the **NULL** value. Otherwise, **value1** is returned. Operations on XML data are supported.

Example:

```
gaussdb=# CREATE TABLE null_if_t1 (
  NI_VALUE1 VARCHAR(10),
  NI_VALUE2 VARCHAR(10)
)DISTRIBUTE BY HASH (NI_VALUE1);

gaussdb=# INSERT INTO null_if_t1 VALUES('abc', 'abc');
gaussdb=# INSERT INTO null_if_t1 VALUES('abc', 'efg');

gaussdb=# SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM null_if_t1 ORDER
BY 1, 2, 3;

ni_value1 | ni_value2 | nullif
-----+-----+-----
abc      | abc      |
abc      | efg      | abc
(2 rows)

gaussdb=# DROP TABLE null_if_t1;
```

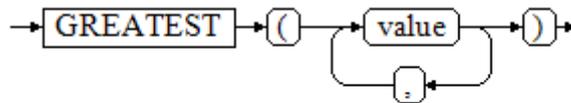
If **value1** is equal to **value2**, **NULL** is returned. Otherwise, **value1** is returned.

```
gaussdb=# SELECT NULLIF('Hello','Hello World');
nullif
-----
Hello
(1 row)
```

- GREATEST (maximum value) and LEAST (minimum value)

Figure 7-5 shows the syntax of a **GREATEST** expression.

Figure 7-5 greatest::=

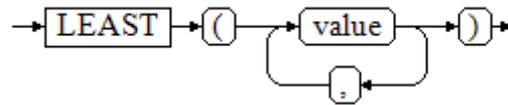


You can select the maximum value from any numerical expression list. Operations on XML data are supported.

```
gaussdb=# SELECT greatest(9000,15555,2.01);
greatest
-----
15555
(1 row)
```

Figure 7-6 shows the syntax of a **LEAST** expression.

Figure 7-6 least::=



You can select the minimum value from any numerical expression list.
Each of the preceding numeric expressions can be converted into a common data type, which will be the data type of the result.
The NULL values in the list will be ignored. The result is **NULL** only if the results of all expressions are **NULL**.
Operations on XML data are supported.

```

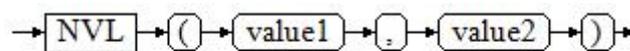
gaussdb=# SELECT least(9000,2);
least
-----
      2
(1 row)
  
```

[Conditional Expression Functions](#) describes the examples.

- NVL

[Figure 7-7](#) shows the syntax of an **NVL** expression.

Figure 7-7 nvl::=



If the value of **value1** is **NULL**, **value2** is returned. Otherwise, **value1** is returned. Operations on XML data are supported.

Example:

```

gaussdb=# SELECT nvl(null,1);
nvl
-----
      1
(1 row)
gaussdb=# SELECT nvl ('Hello World',1);
nvl
-----
Hello World
(1 row)
  
```

7.6.3 Subquery Expressions

Subquery expressions include the following types:

- EXISTS/NOT EXISTS

[Figure 7-8](#) shows the syntax of an **EXISTS/NOT EXISTS** expression.

Figure 7-8 EXISTS/NOT EXISTS::=



The parameter of an **EXISTS** expression is an arbitrary **SELECT** statement, that is, subquery. The subquery is evaluated to determine whether it returns any rows. If it returns at least one row, the result of **EXISTS** is true. If the subquery returns no rows, the result of **EXISTS** is false.

The subquery will generally only be executed long enough to determine whether at least one row is returned, not all the way to completion.

Operations on XML data are not supported.

For example:

```
gaussdb=# CREATE TABLE exists_t1(a int, b int);
gaussdb=# INSERT INTO exists_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE exists_t2(a int, c int);
gaussdb=# INSERT INTO exists_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

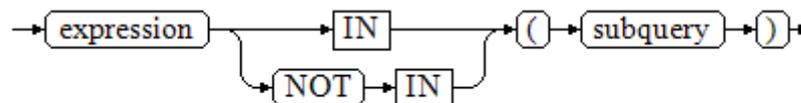
gaussdb=# SELECT * FROM exists_t1 t1 WHERE EXISTS (SELECT * FROM exists_t2 t2 WHERE t2.a =
t1.a);
 a | b
---+---
 3 | 4
 4 | 5
(2 rows)

gaussdb=# DROP TABLE exists_t1, exists_t2;
```

- **IN/NOT IN**

Figure 7-9 shows the syntax of an **IN/NOT IN** expression.

Figure 7-9 IN/NOT IN::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result. The result of **IN** is true if any equal subquery row is found. The result is false if no equal row is found (including the case where the subquery returns no rows).

This is in accordance with SQL normal rules for Boolean combinations of null values. If the columns corresponding to two rows equal and are not empty, the two rows are equal to each other. If any columns corresponding to the two rows do not equal and are not empty, the two rows are not equal to each other. Otherwise, the result is **NULL**. If there are no equal right-hand values and at least one right-hand row yields null, the result of **IN** will be null, not false.

Operations on XML data are not supported.

For example:

```
gaussdb=# CREATE TABLE in_t1(a int, b int);
gaussdb=# INSERT INTO in_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE in_t2(a int, c int);
gaussdb=# INSERT INTO in_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

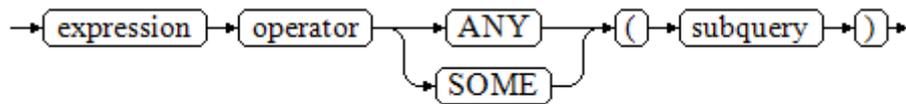
gaussdb=# SELECT * FROM in_t1 t1 WHERE t1.a IN (SELECT t2.a FROM in_t2 t2);
a | b
---+---
3 | 4
4 | 5
(2 rows)

gaussdb=# DROP TABLE in_t1, in_t2;
```

- ANY/SOME

Figure 7-10 shows the syntax of an **ANY/SOME** expression.

Figure 7-10 any/some::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of **ANY** is "true" if any true result is obtained. The result is "false" if no true result is found (including the case where the subquery returns no rows). **SOME** is a synonym of **ANY**. **IN** can be equivalently replaced with **ANY**.

Operations on XML data are not supported.

For example:

```
gaussdb=# CREATE TABLE any_t1(a int, b int);
gaussdb=# INSERT INTO any_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE any_t2(a int, c int);
gaussdb=# INSERT INTO any_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

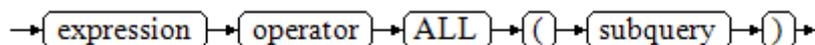
gaussdb=# SELECT * FROM any_t1 t1 WHERE t1.a < ANY(SELECT t2.a FROM any_t2 t2 where t2.a = 3
or t2.a = 4);
a | b
---+---
1 | 2
2 | 3
3 | 4
(3 rows)

gaussdb=# DROP TABLE any_t1, any_t2;
```

- ALL

Figure 7-11 shows the syntax of an **ALL** expression.

Figure 7-11 all::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of ALL is true if all values are true (including the case where the subquery returns no rows). The result is false if any false result is found.

Operations on XML data are not supported.

Example:

```
gaussdb=# CREATE TABLE all_t1(a int, b int);
gaussdb=# INSERT INTO all_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE all_t2(a int, c int);
gaussdb=# INSERT INTO all_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

gaussdb=# SELECT * FROM all_t1 t1 WHERE t1.a < ALL(SELECT t2.a FROM all_t2 t2 where t2.a = 3 or
t2.a = 4);
 a | b
---+---
 1 | 2
 2 | 3
(2 rows)

gaussdb=# DROP TABLE all_t1, all_t2;
```

7.6.4 Array Expressions

IN

expression **IN** (*value* [, ...])

The parentheses on the right contain an expression list. The expression result on the left is compared with the content in the expression list. If the content in the list meets the expression result on the left, the result of **IN** is **true**. If no result meets the requirements, the result of **IN** is **false**.

For example:

```
gaussdb=# SELECT 8000+500 IN (10000, 9000) AS RESULT;
 result
-----
 f
(1 row)
```

NOTE

- If the expression result is null or the expression list does not meet the expression conditions and at least one empty value is returned for the expression list on the right, the result of IN is null rather than **false**. This method is consistent with the Boolean rules used when SQL statements return empty values.
- Operations on XML data are not supported.

NOT IN

expression **NOT IN** (*value* [, ...])

The parentheses on the right contain an expression list. The expression result on the left is compared with the content in the expression list. If the content in the list does not meet the expression result on the left, the result of **NOT IN** is **true**. If any content meets the expression result, the result of **NOT IN** is **false**.

Example:

```
gaussdb=# SELECT 8000+500 NOT IN (10000, 9000) AS RESULT;  
result  
-----  
t  
(1 row)
```

NOTE

- If the query statement result is null or the expression list does not meet the expression conditions and at least one empty value is returned for the expression list on the right, the result of NOT IN is null rather than false. This method is consistent with the Boolean rules used when SQL statements return empty values.
- In all situations, X NOT IN Y equals to NOT(X IN Y).
- Operations on XML data are not supported.

ANY/SOME (array)

expression operator ANY (array expression)

expression operator SOME (array expression)

The right side is a parenthesized expression, which must yield an array value. The result of the expression on the left uses operators to compute and compare the results in each row of the array expression. The comparison result must be a Boolean value.

Example:

```
gaussdb=# SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;  
result  
-----  
t  
(1 row)  
gaussdb=# SELECT 8000+500 < ANY (array[10000,9000]) AS RESULT;  
result  
-----  
t  
(1 row)
```

NOTE

- If at least one comparison result is true, the result of ANY is **true**.
- If no comparison result is true, the result of ANY is **false**.
- If no comparison result is true and the array expression generates at least one null value, the value of ANY is **NULL** rather than **false**. This method is consistent with the Boolean rules used when SQL statements return empty values.
- SOME is a synonym of ANY.
- Operations on XML data are not supported.

ALL (array)

expression operator ALL (array expression)

The right-hand side is a parenthesized expression, which must yield an array value. The result of the expression on the left uses operators to compute and compare the results in each row of the array expression. The comparison result must be a Boolean value.

- The result of ALL is **true** if all comparisons yield true (including the case where the array has zero elements).

- If one or more comparison results are false, the result of ALL is **false**.
- If the array expression generates a null array, the result of ALL is NULL. If the expression on the left yields NULL, the result of ALL is generally NULL (though a non-strict comparison operator could possibly yield a different result). If the array on the right contains any null elements and no false comparison result is found, the result of ALL is NULL, not true (again, assuming a strict comparison operator). This method is consistent with the Boolean rules used when SQL statements return empty values.
- Operations on XML data are not supported.

```
gaussdb=# SELECT 8000+500 < ALL (array[10000,9000]) AS RESULT;  
result  
-----  
t  
(1 row)
```

7.6.5 Row Expressions

Syntax:

row_constructor operator row_constructor

Both sides of the row expression are row constructors. The values of both rows must have the same number of columns and they are compared with each other. The row comparison allows operators including =, <>, <, <=, and >= or a similar operator.

For operators <, <=, >, and >=, the columns in rows are compared from left to right until a pair of columns that are not equal or are empty are detected. If the pair of columns contains at least one null value, the comparison result is null. Otherwise, the comparison result of this pair of columns is the final result. If no unequal or empty column is found, the values in the two rows are equal. The final result is determined based on the operator meaning.

Operations on XML data are not supported.

Example:

```
gaussdb=# SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;  
result  
-----  
t  
(1 row)  
  
gaussdb=# SELECT (4,5,6) > (3,2,1) AS result;  
result  
-----  
t  
(1 row)  
  
gaussdb=# SELECT (4,1,1) > (3,2,1) AS result;  
result  
-----  
t  
(1 row)  
  
gaussdb=# SELECT ('test','data') > ('data','data') AS result;  
result  
-----  
t  
(1 row)  
  
gaussdb=# SELECT (4,1,1) > (3,2,null) AS result;
```

```

result
-----
t
(1 row)

gaussdb=# SELECT (null,1,1) > (3,2,1) AS result;
result
-----
(1 row)

gaussdb=# SELECT (null,5,6) > (null,5,6) AS result;
result
-----
(1 row)

gaussdb=# SELECT (4,5,6) > (4,5,6) AS result;
result
-----
f
(1 row)

gaussdb=# SELECT (2,2,5) >= (2,2,3) AS result;
result
-----
t
(1 row)

gaussdb=# SELECT (2,2,1) <= (2,2,3) AS result;
result
-----
t
(1 row)

```

The use of operators = and <> is slightly different from other operators. If all columns in the two rows are not empty and meet the operator condition, the two rows meet the operator condition. If any column in the two rows is not empty and does not meet the operator condition, the two rows do not meet the operator condition. If any column in the two rows is empty, the comparison result is null.

For example:

```

gaussdb=# SELECT (1,2,3) = (1,2,3) AS result;
result
-----
t
(1 row)

gaussdb=# SELECT (1,2,3) <> (2,2,3) AS result;
result
-----
t
(1 row)

gaussdb=# SELECT (2,2,3) <> (2,2,null) AS result;
result
-----
(1 row)

gaussdb=# SELECT (null,5,6) <> (null,5,6) AS result;
result
-----
(1 row)

```

7.7 Pseudocolumn

ROWNUM is a pseudocolumn that returns a number indicating the row number of the obtained query result. The value of **ROWNUM** in the first row is **1**, the value of **ROWNUM** in the second row is **2**, and so on.

ROWNUM can be used to limit the number of rows returned by a query, as shown in the following example:

```
gaussdb=# CREATE TABLE Students (name varchar(20), id int) with (STORAGE_TYPE = USTORE);
gaussdb=# INSERT INTO Students VALUES ('Jack', 35);
gaussdb=# INSERT INTO Students VALUES ('Leon', 15);
gaussdb=# INSERT INTO Students VALUES ('James', 24);
gaussdb=# INSERT INTO Students VALUES ('Taker', 81);
gaussdb=# INSERT INTO Students VALUES ('Mary', 25);
gaussdb=# INSERT INTO Students VALUES ('Rose', 64);
gaussdb=# INSERT INTO Students VALUES ('Perl', 18);
gaussdb=# INSERT INTO Students VALUES ('Under', 57);
gaussdb=# INSERT INTO Students VALUES ('Angel', 101);
gaussdb=# INSERT INTO Students VALUES ('Frank', 20);
gaussdb=# INSERT INTO Students VALUES ('Charlie', 40);
```

-- Output the first 10 rows of data records in the **Students** table.

```
gaussdb=# SELECT * FROM Students WHERE rownum <= 10;
```

```
name | id
-----+-----
Jack | 35
Leon | 15
James | 24
Taker | 81
Mary | 25
Rose | 64
Perl | 18
Under | 57
Angel | 101
Frank | 20
(10 rows)
```

If the statement has a clause, the output rows are reordered according to the clause.

```
gaussdb=# SELECT * FROM Students WHERE rownum < 5 order by 1;
```

```
name | id
-----+-----
Jack | 35
James | 24
Leon | 15
Taker | 81
(4 rows)
```

If a subquery has a clause but the condition is placed in the outermost query, you can use the **ROWNUM** condition after sorting.

```
gaussdb=# SELECT rownum, * FROM (SELECT * FROM Students order by 1) WHERE rownum <= 2;
```

```
rownum | name | id
-----+-----+-----
1 | Angel | 101
2 | Charlie | 40
(2 rows)
```

As long as **ROWNUM** is greater than a specific positive integer, the condition is always false. As shown in the following example, the statement does not return any result in the table.

```
gaussdb=# SELECT * FROM Students WHERE rownum > 1;
```

```
name | id
-----+-----
(0 rows)
```

Use **ROWNUM** to assign a value to each row within a certain range of the table.

```
gaussdb=# SELECT * FROM Students;
 name | id
-----+---
 Jack | 35
 Leon | 15
 James | 24
 Taker | 81
 Mary | 25
 Rose | 64
 Perl | 18
 Under | 57
 Angel | 101
 Frank | 20
 Charlie | 40
(11 rows)

gaussdb=# UPDATE Students set id = id + 5 WHERE rownum < 4;
UPDATE 3
gaussdb=# SELECT * FROM Students;
 name | id
-----+---
 Jack | 40
 Leon | 20
 James | 29
 Taker | 81
 Mary | 25
 Rose | 64
 Perl | 18
 Under | 57
 Angel | 101
 Frank | 20
 Charlie | 40
(11 rows)

gaussdb=# DROP TABLE Students;
DROP TABLE
```

The restrictions on using **ROWNUM** are as follows:

- Do not use ROWNUM as an alias to avoid ambiguity in SQL statements.
- Do not use ROWNUM when creating an index.
- Do not use ROWNUM as the default value when creating a table.
- Do not use ROWNUM as an alias in the WHERE clause.
- Do not use ROWNUM when inserting data.
- Do not use ROWNUM in a tableless query.
- Do not use ROWNUM in the LIMIT clause.
- Do not use ROWNUM as a parameter of the EXECUTE statement.
- Do not use ROWNUM to update a clause in the UPSERT statement.
- If a subquery contains UNION ALL and the parent query contains ROWNUM, the subquery cannot be pulled up.
- Do not use ROWNUM as a projection column or WHERE condition in the SELECT ... FOR UPDATE statement.
- If the HAVING clause contains ROWNUM (not in an aggregate function), the GROUP BY clause must also contain ROWNUM (not in an aggregate function), unless the GROUP BY clause contains an expression, for example, **SELECT a + a FROM t group by a + a having rownum < 5**.
- If the ROWNUM condition exists in the HAVING clause, the HAVING clause cannot be pushed down to any scan node.

```
gaussdb=# CREATE TABLE test (a int, b int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO test SELECT generate_series, generate_series FROM generate_series(1, 10);
INSERT 0 10
```

-- The rownum condition cannot be pushed down to **seqscan**.

```
gaussdb=# EXPLAIN SELECT a,rownum FROM test group by a,rownum having rownum < 5;
QUERY PLAN
```

```
-----
HashAggregate (cost=3.45..3.49 rows=3 width=4)
  Group By Key: a, ROWNUM
  Filter: ((ROWNUM) < 5::numeric)
  -> Rownum (cost=0.19..3.40 rows=10 width=4)
      -> Streaming (type: GATHER) (cost=0.19..3.40 rows=10 width=4)
          Node/s: All datanodes
          -> Seq Scan on test (cost=0.00..3.03 rows=10 width=4)
(7 rows)
```

- If a subquery contains the ROWNUM condition, the predicate cannot be pushed down to any scan node.

```
gaussdb=# EXPLAIN SELECT * FROM (SELECT * FROM test WHERE rownum < 5) WHERE b < 5; -- The
b<5 WHERE clause cannot be pushed down to Seq Scan.
QUERY PLAN
```

```
-----
Streaming (type: GATHER) (cost=0.06..0.76 rows=2 width=8)
  Node/s: All datanodes
  -> Subquery Scan on __unnamed_subquery__ (cost=0.00..0.63 rows=2 width=8)
      Filter: (__unnamed_subquery__.b < 5)
      -> Rownum (cost=0.00..0.57 rows=5 width=8)
          StopKey: (ROWNUM < 5::numeric)
          -> Streaming(type: BROADCAST) (cost=0.00..1.71 rows=15 width=8)
              Spawn on: All datanodes
              -> Rownum (cost=0.00..1.68 rows=5 width=8)
                  StopKey: (ROWNUM < 5::numeric)
                  -> Seq Scan on test (cost=0.00..3.03 rows=10 width=8)
(11 rows)
```

```
gaussdb=# DROP TABLE test;
DROP TABLE
```

CAUTION

- The query result of **ROWNUM** may be unstable due to the sequence in which the CN receives DN data.

```
gaussdb=# CREATE TABLE test(a int, b int);
gaussdb=# INSERT INTO test VALUES(generate_series(1,10),generate_series(1,10));
```

```
-- Output the first 10 rows of data records in the Students table.
```

```
gaussdb=# SELECT rownum,* FROM test;
rownum | a | b
```

```
-----+-----+-----
1 | 1 | 1
2 | 2 | 2
3 | 5 | 5
4 | 6 | 6
5 | 8 | 8
6 | 9 | 9
7 | 3 | 3
8 | 4 | 4
9 | 7 | 7
10 | 10 | 10
(10 rows)
```

```
-- The CN receives data from DN2 and then from DN1.
```

```
gaussdb=# SELECT rownum,* FROM test;
rownum | a | b
```

```
-----+-----+-----
1 | 3 | 3
2 | 4 | 4
3 | 7 | 7
4 | 10 | 10
5 | 1 | 1
6 | 2 | 2
7 | 5 | 5
8 | 6 | 6
9 | 8 | 8
10 | 9 | 9
(10 rows)
```

- It is not recommended that the **ROWNUM** condition be used in the **JOIN ON** clause. In GaussDB, when the **ROWNUM** condition is used in the **JOIN ON** clause, the behavior in the **LEFT JOIN**, **RIGHT JOIN**, **FULL JOIN**, and **MERGE INTO** scenarios is different from that in other databases, causing risks in service migration.

7.8 Type Conversion

7.8.1 Overview

Context

SQL is a typed language. That is, every data item has an associated data type which determines its behavior and allowed usage. GaussDB has an extensible type system that is more general and flexible than other SQL implementations. Hence, most type conversion behaviors in GaussDB are governed by general rules. This allows the use of mixed-type expressions.

The GaussDB scanner/parser divides lexical elements into five fundamental categories: integers, floating-point numbers, strings, identifiers, and keywords.

Constants of most non-numeric types are first classified as strings. The SQL language definition allows specifying type names with constant strings. The following is an example:

```
gaussdb=# SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
label | value
-----+-----
Origin | (0,0)
(1 row)
```

In this example, there are two literal constants of the text and point types, respectively. If a type is not specified for a string literal, then the placeholder type unknown is assigned initially.

There are four fundamental SQL constructs requiring distinct type conversion rules in GaussDB parser:

- **Function calls**
Much of the SQL type system is built around a rich set of functions. Functions can have one or more arguments. Since SQL permits function overloading, the function name alone does not uniquely identify the function to be called. The parser must select the right function based on the data types of the supplied arguments.
- **Operators**
SQL allows expressions with prefix and postfix unary (one-argument) operators, as well as binary (two-argument) operators. Like functions, operators can be overloaded, so the same problem of selecting the right operator exists.
- **Value storage**
The INSERT and UPDATE statements place the results of expressions into a table. The expressions in the statement must be matched up with, and perhaps converted to, the types of the target columns.
- **UNION, CASE, and Related Constructs**
Since all query results from a unionized SELECT statement must appear in a single set of columns, the types of the results of each SELECT clause must be matched up and converted to a uniform set. Similarly, the result expressions of a CASE construct must be converted to a common type so that the CASE expression as a whole has a known output type. The same holds for ARRAY constructs, and for the GREATEST and LEAST functions.

The system catalog `pg_cast` stores information about which conversions, or casts, exist between which data types, and how to perform those conversions. For details, see [PG_CAST](#).

The return type and conversion behavior of an expression are determined during semantic analysis. Data types are divided into several basic type categories, including Boolean, numeric, string, bitstring, datetime, timespan, geometric, and network. Within each category there can be one or more preferred types, which are preferred when there is a choice of possible types. With careful selection of preferred types and available implicit casts, it is possible to ensure that ambiguous expressions (those with multiple candidate parsing solutions) can be resolved in a useful way.

All type conversion rules are designed based on the following principles:

- Implicit conversions should never have surprising or unpredictable outcomes.
- There should be no extra overhead in the parser or executor if a query does not need implicit type conversion. That is, if a query is well-formed and the types already match, then the query should execute without spending extra time in the parser and without introducing unnecessary implicit conversion calls in the query.
- Additionally, if a query usually requires an implicit conversion for a function, and if then the user defines a new function with the correct argument types, the parser should use this new function.
- XML data does not support implicit type conversion, including implicit conversion between strings and XML types.

7.8.2 Operators

Operator Type Resolution

1. Select the operators to be considered from the `pg_operator` system catalog. Considered operators are those with the matching name and argument count. If the search path finds multiple available operators, only the most suitable one is considered.
2. Look for the best match.
 - a. Discard candidate operators for which the input types do not match and cannot be converted (using an implicit conversion) to match. Unknown text can be converted to anything for this purpose. If only one candidate remains, use it; else continue to the next step.
 - b. Run through all candidates and keep those with the most exact matches on input types. In this case, the domain types are considered the same as their basic types. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.
 - c. Run through all candidates and keep those that accept preferred types (of the input data type's type category) at the most positions where type conversion will be required. Keep all candidates if none accepts preferred types. If only one candidate remains, use it; else continue to the next step.
 - d. If any input arguments are of unknown types, check the type categories accepted at those argument positions by the remaining candidates. At each position, select the string category if any candidate accepts that category. (This bias towards string is appropriate since an unknown-type literal looks like a string.) Otherwise, if all the remaining candidates accept the same type category, select that category; otherwise fail because the correct choice cannot be deduced without more clues. Now discard candidates that do not accept the selected type category. Furthermore, if any candidate accepts a preferred type in that category, discard candidates that accept non-preferred types for that argument. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.
 - e. If there are both unknown and known-type arguments, and all the known-type arguments have the same type, assume that the unknown arguments are also of that type, and check which candidates can accept that type at the unknown-argument positions. If exactly one candidate passes this test, use it. Otherwise, an error occurs.

Examples

Example 1: Use factorial operator type resolution. There is only one factorial operator (postfix !) defined in the system catalog, and it takes an argument of bigint type. The scanner assigns bigint as an initial type to the argument in this query expression:

```
gaussdb=# SELECT 40 ! AS "40 factorial";
      40 factorial
-----
815915283247897734345611269596115894272000000000
(1 row)
```

The parser performs a type conversion on the operand and the query is equivalent to:

```
gaussdb=# SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

Example 2: String concatenation operator type resolution. A string-like syntax is used for working with string types and for working with complex extension types. Strings with unspecified type are matched with likely operator candidates. An example with one unspecified argument:

```
gaussdb=# SELECT text 'abc' || 'def' AS "text and unknown";
text and unknown
-----
abcdef
(1 row)
```

In this example, the parser looks for an operator whose parameters are of the text type. Such an operator is found.

Here is a concatenation of two values of unspecified types:

```
gaussdb=# SELECT 'abc' || 'def' AS "unspecified";
unspecified
-----
abcdef
(1 row)
```

NOTE

In this case there is no initial hint for which type to use, since no types are specified in the query. So, the parser looks for all candidate operators and finds that there are candidates accepting both string-category and bit-string-category inputs. Since string category is preferred when available, that category is selected, and then the preferred type for strings, text, is used as the specific type to resolve the unknown-type literals as.

Example 3: Absolute-value and negation operator type resolution. The GaussDB operator catalog has several entries for the prefix operator @. All the entries implement absolute-value operations for various numeric data types. One of these entries is for the float8 type, which is the preferred type in the numeric category. Therefore, GaussDB will use that entry when faced with an unknown input:

```
gaussdb=# SELECT @ '-4.5' AS "abs";
abs
----
4.5
(1 row)
```

The system has implicitly resolved the unknown-type literal as type float8 before applying the chosen operator.

Example 4: Use the array inclusion operator type resolution as an example. Here is another example of resolving an operator with one known and one unknown input:

```
gaussdb=# SELECT array[1,2] <@ '{1,2,3}' as "is subset";
is subset
-----
t
(1 row)
```

NOTE

The GaussDB operator catalog has several entries for the infix operator <@, but the only two that could possibly accept an integer array on the left side are array inclusion (anyarray <@ anyarray) and range inclusion (anyelement <@ anyrange). Since none of these polymorphic pseudo-types (see section [Pseudo-Types](#)) is considered preferred, the parser cannot resolve the ambiguity on that basis. However, the last resolution rule tells it to assume that the unknown-type literal is of the same type as the other input, that is, integer array. Now only one of the two operators can match, so array inclusion is selected. (Had range inclusion been selected, we would have gotten an error, because the string does not have the right format to be a range literal.)

7.8.3 Functions

Function Type Resolution

1. Select the functions to be considered from the pg_proc system catalog. If a non-schema-qualified function name was used, the functions in the current search path are considered. If a qualified function name was given, only functions in the specified schema are considered.
If the search path finds multiple functions of different argument types, a proper function in the path is considered.
2. Check for a function accepting exactly the input argument types. If the function exists, use it. Cases involving unknown types will never find a match at this step.
3. If no exact match is found, see if the function call appears to be a special type conversion request.
4. Look for the best match.
 - a. Discard candidate functions for which the input types do not match and cannot be converted (using an implicit conversion) to match. Unknown text can be converted to anything for this purpose. If only one candidate remains, use it; else continue to the next step.
 - b. Run through all candidates and keep those with the most exact matches on input types. Domains are considered the same as their base type for this purpose. Keep all candidates if none has exact matches. If only one candidate remains, use it; else continue to the next step.
 - c. Run through all candidates and keep those that accept preferred types at the most positions where type conversion will be required. Keep all candidates if none accepts preferred types. If only one candidate remains, use it; else continue to the next step.
 - d. If any input arguments are of unknown types, check the type categories accepted at those argument positions by the remaining candidates. At each position, select the string category if any candidate accepts that category. (This bias towards string is appropriate since an unknown-type

literal looks like a string.) Otherwise, if all the remaining candidates accept the same type category, select that category; otherwise fail because the correct choice cannot be deduced without more clues. Now discard candidates that do not accept the selected type category. Furthermore, if any candidate accepts a preferred type in that category, discard candidates that accept non-preferred types for that argument. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.

- e. If there are both unknown and known arguments, and all the known arguments have the same type, assume that the unknown arguments are also of that type, and check which candidates can accept that type at the unknown-argument positions. If exactly one candidate passes this test, use it. Otherwise, fail.

Examples

Example 1: Use the rounding function argument type resolution as the first example. There is only one round function that takes two arguments; it takes a first argument of type numeric and a second argument of type integer. In this case, the following query automatically converts the first argument of type integer to numeric:

```
gaussdb=# SELECT round(4, 4);
round
-----
4.0000
(1 row)
```

That query is actually transformed by the parser to:

```
gaussdb=# SELECT round(CAST (4 AS numeric), 4);
```

Since numeric constants with decimal points are initially assigned the numeric type, the following query will require no type conversion and therefore might be slightly more efficient:

```
gaussdb=# SELECT round(4.0, 4);
```

Example 2: Use the substring function type resolution as the second example. There are several substr functions, one of which takes types text and integer. If the function is called with a string constant of unspecified type, the system chooses the candidate function that accepts an argument of the preferred category string (namely of type text).

```
gaussdb=# SELECT substr('1234', 3);
substr
-----
34
(1 row)
```

If the string is declared to be of varchar type, as might be the case if it comes from a table, then the parser will try to convert it to become text:

```
gaussdb=# SELECT substr(varchar '1234', 3);
substr
-----
34
(1 row)
```

This is transformed by the parser to effectively become:

```
gaussdb=# SELECT substr(CAST (varchar '1234' AS text), 3);
```

NOTE

The parser learns from the `pg_cast` catalog that text and varchar are binary-compatible, meaning that one can be passed to a function that accepts the other without doing any physical conversion. Therefore, no type conversion is really inserted in this case.

And, if the function is called with an argument of integer type, the parser will try to convert that to text:

```
gaussdb=# SELECT substr(1234, 3);
substr
-----
34
(1 row)
```

This is transformed by the parser to effectively become:

```
gaussdb=# SELECT substr(CAST (1234 AS text), 3);
substr
-----
34
(1 row)
```

7.8.4 Value Storage

Value Storage Type Resolution

1. Search for an exact match with the target column.
2. Try to convert the expression to the target type. This will succeed if there is a registered cast between the two types. If the expression is an unknown-type literal, the content of the literal string will be fed to the input conversion routine for the target type.
3. Check to see if there is a sizing cast for the target type. A sizing cast is a cast from that type to itself. If one is found in the `pg_cast` system catalog, apply it to the expression before storing into the destination column. The implementation function for such a cast always takes an extra parameter of type **integer**. The parameter receives the destination column's **atttypmod** value (typically its declared length, although the interpretation of **atttypmod** varies for different data types), and may take a third Boolean parameter that says whether the cast is explicit or implicit. The cast function is responsible for applying any length-dependent semantics such as size checking or truncation.

Examples

Use the character storage type conversion as an example. For a target column declared as `character(20)`, the following statement shows that the stored value is sized correctly:

```
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# CREATE TABLE tpcds.value_storage_t1 (
VS_COL1 CHARACTER(20)
)DISTRIBUTE BY HASH (VS_COL1);
gaussdb=# INSERT INTO tpcds.value_storage_t1 VALUES('abcdef');
gaussdb=# SELECT VS_COL1, octet_length(VS_COL1) FROM tpcds.value_storage_t1;
vs_col1      | octet_length
-----+-----
abcdef       |          20
(1 row)
```

```
)  
gaussdb=# DROP TABLE tpcds.value_storage_t1;  
gaussdb=# DROP SCHEMA tpcds;
```

NOTE

What has happened here is that the two unknown literals are resolved to text by default, allowing the || operator to be resolved as text concatenation. Then the text result of the operator is converted to bpchar ("blank-padded char", the internal name of the character data type) to match the target column type. Since the conversion from text to bpchar is binary-coercible, this conversion does not insert any real function call. Finally, the sizing function bpchar(bpchar, integer, Boolean) is found in the system catalog and used for the operator's result and the stored column length. This type-specific function performs the required length check and addition of padding spaces.

7.8.5 UNION, CASE, and Related Constructs

SQL UNION constructs must match up possibly dissimilar types to become a single result set. The resolution algorithm is applied separately to each output column of a union query. INTERSECT and EXCEPT constructs resolve dissimilar types in the same way as UNION. The CASE, ARRAY, VALUES, GREATEST and LEAST constructs use the identical algorithm to match up their component expressions and select a result data type.

Type Resolution for UNION, CASE, and Related Constructs

- If all inputs are of the same type and are not unknown, resolve them as the unknown type.
- If all inputs are of the unknown type, resolve them as the text type (the preferred type of the string category). Otherwise, unknown inputs are ignored. (Exception: The UNION operation resolves a group of two unknown types into the text type, and then continues to match the type with other groups.)
- If the inputs are not all of the same type category, a failure will be resulted. (The unknown type is not included in this case.)
- If the inputs are all of the same type category, choose the top preferred type in that category. (Exception: The UNION operation regards the type of the first branch as the selected type.)

NOTE

- **typcategory** in the pg_type system catalog indicates the data type category. **typispreferred** indicates whether a type is preferred in **typcategory**.
- Convert all inputs to the selected type. (Retain the original lengths of strings). Fail if there is not an implicit conversion from a given input to the selected type.
- If the input contains the json, txid_snapshot, sys_refcursor, or geometry type, UNION cannot be performed.

Type Resolution for CASE and COALESCE in TD Compatibility Type

- If all inputs are of the same type and are not unknown, resolve them as the unknown type.
- If all inputs are of the unknown type, resolve them as the text type.

- If inputs are of the string type (including unknown which is resolved as text) and digit type, resolve them as the string type. If the inputs are not of the two types, an error will be reported.
- If the inputs are all of the same type category, choose the top preferred type in that category.
- Convert all inputs to the selected type. Fail if there is not an implicit conversion from a given input to the selected type.

Type Resolution for CASE in ORA Compatibility Mode

decode(expr, search1, result1, search2, result2, ..., defresult): When the **sql_beta_feature** is set to **a_style_coerce**, the final return value type of the expression is set to the data type of result1 or a higher-precision data type in the same type as result1, as that in ORA-compatible mode. (For example, numeric and int are both numeric data types, but numeric has higher precision and priority than int.) For CASE WHEN, the behavior is the same as the default behavior in ORA-compatible mode.

- If all inputs are of the same type and are not unknown, resolve them as the unknown type. Otherwise, proceed to the next step.
- Set the data type of result1 to the final return value type **preferType**, which belongs to **preferCategory**.
- Consider the data types of result2, result3, and defresult in sequence. If the type category is also **preferCategory**, which is the same as that of result1, check whether the precision (priority) is higher than that of **preferType**. If it is, update **preferType** to a data type with higher precision. If the type category is not **preferCategory**, check whether the category can be implicitly converted to **preferType**. If it cannot, an error is reported.
- Uses the data type recorded by **preferType** as the return value type of the expression. The expression result is implicitly converted to this data type.

Note 1:

There is a special case where the character type of a super-large number is converted to the numeric type, for example, **select decode(1, 2, 2, '53465465676465454657567678676')**, in which the large number exceeds the range of the bigint and double types. If result1 is of the numeric type and does not meet the condition that all inputs are of the same type, the type of the return value is set to numeric to be compatible with this special case.

Note 2:

Priority of the numeric types: numeric > float8 > float4 > int8 > int4 > int2 > int1

Priority of the character types: text > varchar (nvarchar2) > bpchar > char

Priority of date types: timestamptz > timestamp > smalldatetime > date > abstime > timetz > time

Priority of date span types: interval > tinterval > reltime

Note 3:

The following figure shows the supported implicit type conversion when **set sql_beta_feature** is set to '**a_style_coerce**' in ORA-compatible mode. \ indicates

that conversion is not required, **yes** indicates that conversion is supported, and the null value indicates that conversion is not supported.

	bool	int1	int2	int4	int8	float4	float8	numeric	money	char	bpchar	varchar2	nvarchar2	text/clob	raw	blob	date	time	timetz	timestamp	timestampz	smalldatetime	interval	reftime	abstime
bool	\																								
int1	yes	\	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes											
int2		yes	\	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes											
int4		yes	yes	\	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes											
int8	yes	yes	yes	yes	\	yes	yes	yes	yes	yes	yes	yes	yes	yes											
float4	yes	yes	yes	yes	yes	\	yes	yes	yes	yes	yes	yes	yes	yes											
float8	yes	yes	yes	yes	yes	yes	\	yes	yes	yes	yes	yes	yes	yes											
numeric	yes	yes	yes	yes	yes	yes	yes	\		yes	yes	yes	yes	yes											
money									\																
char	yes	yes	yes	yes	yes	yes	yes	yes		\	yes	yes	yes	yes											
bpchar	yes	yes	yes	yes	yes	yes	yes	yes		yes	\	yes	yes	yes											
varchar2	yes	yes	yes	yes	yes	yes	yes	yes		yes	yes	\	yes	yes	yes										
nvarchar2	yes	yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	\	yes											
text/clob	yes	yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	\											
raw												yes	yes	yes	\	yes									
blob														yes	\										
date												yes	yes	yes	yes		\			yes	yes	yes			yes
time														yes			\	yes							
timetz														yes			yes	\							
timestamp												yes	yes	yes	yes	yes			\		yes	yes			yes
timestampz														yes			yes			yes	\	yes			yes
smalldatetime												yes		yes			yes			yes	yes	\			yes
interval												yes	yes	yes									\	yes	
reftime														yes									yes	\	
abstime														yes		yes				yes	yes	yes			\

Examples

Example 1: Use type resolution with underspecified types in a union as the first example. Here, the unknown-type literal 'b' will be resolved to the text type.

```
gaussdb=# SELECT text 'a' AS "text" UNION SELECT 'b';
text
-----
a
b
(2 rows)
```

Example 2: Use type resolution in a simple union as the second example. The literal 1.2 is of numeric type, and the integer value 1 can be cast implicitly to numeric type, so that type is used.

```
gaussdb=# SELECT 1.2 AS "numeric" UNION SELECT 1;
numeric
-----
1
1.2
(2 rows)
```

Example 3: Use type resolution in a transposed union as the third example. Since the real type cannot be implicitly cast to integer, but integer can be implicitly cast to real, the union result type is resolved as real.

```
gaussdb=# SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
real
-----
1
2.2
(2 rows)
```

Example 4: In TD mode, if input parameters for **COALESCE** are of int and varchar types, resolve them as the varchar type. In ORA mode, an error is reported.

```
-- In Oracle mode, create the oracle_1 database compatible with Oracle.
gaussdb=# CREATE DATABASE oracle_1 dbcompatibility = 'ORA';

-- Switch to the oracle_1 database.
gaussdb=# \c oracle_1

-- Create the t1 table.
oracle_1=# CREATE TABLE t1(a int, b varchar(10));
-- View the execution plan of the query statement whose coalesce parameter is of the int or varchar type.
a_1=# EXPLAIN SELECT coalesce(a, b) FROM t1;
ERROR: COALESCE types integer and character varying cannot be matched
LINE 1: EXPLAIN SELECT coalesce(a, b) FROM t1;
                        ^
CONTEXT: referenced column: coalesce
```

```
-- Delete the table.
oracle_1=# DROP TABLE t1;

-- Switch to the testdb database.
oracle_1=# \c testdb

-- In TD mode, create the td_1 database compatible with Teradata.
gaussdb=# CREATE DATABASE td_1 dbcompatibility = 'TD';

-- Switch to the td_1 database.
gaussdb=# \c td_1

-- Create the t2 table.
td_1=# CREATE TABLE t2(a int, b varchar(10));

-- View the execution plan of the query statement whose coalesce parameter is of the int or varchar type.
td_1=# EXPLAIN VERBOSE select coalesce(a, b) from t2;
          QUERY PLAN
-----
Data Node Scan (cost=0.00..0.00 rows=0 width=0)
  Output: (COALESCE((t2.a)::character varying, t2.b))
  Node/s: All datanodes
  Remote query: SELECT COALESCE(a::character varying, b) AS "coalesce" FROM public.t2
(4 rows)

-- Delete the table.
td_1=# DROP TABLE t2;

-- Switch to the testdb database.
td_1=# \c testdb

-- Delete Oracle- and TD-compatible databases.
gaussdb=# DROP DATABASE oracle_1;
gaussdb=# DROP DATABASE td_1;
```

Example 5: In ORA mode, set the final return value type of the expression to the data type of result1 or a higher-precision data type whose category is the same as that of the data type of result1.

```
-- In ORA mode, create the ora_1 database compatible with ORA.
gaussdb=# CREATE DATABASE ora_1 dbcompatibility = 'A';

-- Switch to the ora_1 database.
gaussdb=# \c ora_1

-- Enable the decode compatibility parameters.
set sql_beta_feature='a_style_coerce';

-- Create the t1 table.
ora_1=# CREATE TABLE t1(c_int int, c_float8 float8, c_char char(10), c_text text, c_date date);

-- Insert data.
ora_1=# INSERT INTO t1 VALUES(1, 2, '3', '4', date '12-10-2010');
```

```
-- The data type of result1 is char and that of defresult is text. The precision of text is higher, and the type
of the return value is changed to text from char.
ora_1=# SELECT decode(1, 2, c_char, c_text) AS result, pg_typeof(result) FROM t1;
result | pg_typeof
-----+-----
4      | text
(1 row)

-- The data type of result1 is int, which is a numeric type. The type of the return value is set to numeric.
ora_1=# SELECT decode(1, 2, c_int, c_float8) AS result, pg_typeof(result) FROM t1;
result | pg_typeof
-----+-----
2      | numeric
(1 row)

-- The implicit conversion from the data type of defresult to that of result1 does not exist. If it is performed,
an error is reported.
ora_1=# SELECT decode(1, 2, c_int, c_date) FROM t1;
ERROR: CASE types integer and timestamp without time zone cannot be matched
LINE 1: SELECT decode(1, 2, c_int, c_date) FROM t1;
                        ^
CONTEXT: referenced column: c_date

-- Disable the decode compatibility parameters.
set sql_beta_feature='none';

-- Delete the table.
ora_1=# DROP TABLE t1;
DROP TABLE

-- Switch to the testdb database.
ora_1=# \c testdb

-- Delete the ORA-compatible database.
gaussdb=# DROP DATABASE ora_1;
DROP DATABASE
```

Example 6: The UNION operation resolves a group of two unknown types into the text type, and then continues to match the type with other groups.

```
-- Resolve the first two NULL values of the unknown type as the text type. Then, match the text type with
the third element of the varchar2 type, and select the text type.
gaussdb=# SELECT "text", pg_typeof("text") as type from (SELECT NULL AS "text" UNION ALL SELECT
NULL AS "text" UNION ALL SELECT 'a'::varchar2 as "text");
text | type
-----+-----
| text
| text
a    | text
(3 rows)
```

7.9 System Operation

GaussDB text runs SQL statements to perform different system operations, such as setting variables, displaying the execution plan, and collecting garbage data.

Setting Variables

For details about how to set various parameters for a session or transaction, see [SET](#).

Displaying the Execution Plan

For details about how to display the execution plan that GaussDB makes for SQL statements, see [EXPLAIN](#).

Specifying a Checkpoint in Transaction Logs

By default, WALs periodically specify checkpoints in a transaction log. **CHECKPOINT** forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system. For details, see [CHECKPOINT](#).

Collecting Unnecessary Data

For details about how to collect garbage data and analyze a database as required, For details, see [VACUUM](#).

Collecting Statistics

For details about how to collect statistics on tables in databases, see [ANALYZE | ANALYZE](#).

Setting the Constraint Check Mode for the Current Transaction

For details about how to set the constraint check mode for the current transaction, see [SET CONSTRAINTS](#).

7.10 Controlling Transactions

A transaction is a user-defined sequence of database operations, which form an integral unit of work.

Starting a Transaction

GaussDB starts a transaction using **START TRANSACTION** and **BEGIN**. For details, see [START TRANSACTION](#) and [BEGIN](#).

Setting a Transaction

GaussDB sets a transaction using **SET TRANSACTION** or **SET LOCAL TRANSACTION**. For details, see [SET TRANSACTION](#).

Committing a Transaction

GaussDB commits all operations of a transaction using **COMMIT** or **END**. For details, see [COMMIT | END](#).

Rolling Back a Transaction

If a fault occurs during a transaction and the transaction cannot proceed, the system performs rollback to cancel all the completed database operations related to the transaction. For details, see [ROLLBACK](#).

 NOTE

If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. If one of the statements fails, the entire request will be rolled back.

7.11 DDL Syntax Overview

Data definition language (DDL) is used to define or modify an object in a database, such as a table, an index, or a view.

 NOTE

GaussDB does not support DDL if its CN is unavailable. For example, if a CN in cluster is faulty, creating a database or a table will fail.

Defining a CMK

Client master keys (CMKs) are used to encrypt column encryption keys (CEKs) for the encrypted database feature. CMK definition includes creating and deleting a CMK. For details about related SQL statements, see [Table 7-143](#).

Table 7-143 SQL statements for defining a CMK

Function	SQL Statement
Creating a CMK	CREATE CLIENT MASTER KEY
Deleting a CMK	DROP CLIENT MASTER KEY

Defining a CEK

CEKs are used to encrypt data for the encrypted database feature. You can create a CEK, change the client master key specified by a CEK, and delete a CEK. For details about related SQL statements, see [Table 7-143](#).

Table 7-144 SQL statements for defining a CEK

Function	SQL Statement
Creating a CEK	CREATE COLUMN ENCRYPTION KEY
Changing the client master key specified by a CEK	7.14.173-ALTER COLUMN ENCRYPTION KEY
Deleting a CEK	DROP COLUMN ENCRYPTION KEY

Defining a Database

A database is the warehouse for organizing, storing, and managing data. Defining a database includes: creating a database, altering the database attributes, and dropping the database. For details about related SQL statements, see [Table 7-145](#).

Table 7-145 SQL statements for defining a database

Function	SQL Statement
Creating a database	CREATE DATABASE
Altering database attributes	ALTER DATABASE
Dropping a Database	DROP DATABASE

Defining a Schema

A schema is the set of a group of database objects and is used to control the access to the database objects. For details about related SQL statements, see [Table 7-146](#).

Table 7-146 SQL statements for defining a schema

Function	SQL Statement
Creating a schema	CREATE SCHEMA
Altering schema attributes	ALTER SCHEMA
Dropping a schema	DROP SCHEMA

Defining a Tablespace

A tablespace is used to manage data objects and corresponds to a catalog on a disk. For details about related SQL statements, see [Table 7-147](#).

Table 7-147 SQL statements for defining a tablespace

Function	SQL Statement
Creating a tablespace	CREATE TABLESPACE
Altering tablespace attributes	ALTER TABLESPACE
Dropping a tablespace	DROP TABLESPACE

Defining a Table

A table is a special data structure in a database and is used to store data objects and relationship between data objects. For details about related SQL statements, see [Table 7-148](#).

Table 7-148 SQL statements for defining a table

Function	SQL Statement
Creating a table	CREATE TABLE
Altering table attributes	ALTER TABLE
Dropping a table	DROP TABLE

Defining a Partitioned Table

A partitioned table is a logical table used to improve query performance and does not store data (data is stored in ordinary tables). For details about related SQL statements, see [Table 7-149](#).

Table 7-149 SQL statements for defining a partitioned table

Function	SQL Statement
Creating a partitioned table	CREATE TABLE PARTITION
Create a partition	ALTER TABLE PARTITION
Altering partitioned table attributes	ALTER TABLE PARTITION
Deleting a partition	ALTER TABLE PARTITION
Dropping a partitioned table	DROP TABLE

Defining an Index

An index indicates the sequence of values in one or more columns in a database table. It is a data structure that improves the speed of data access to specific information in a database table. For details about related SQL statements, see [Table 7-150](#).

Table 7-150 SQL statements for defining an index

Function	SQL Statement
Creating an index	CREATE INDEX
Altering index attributes	ALTER INDEX
Dropping an index	DROP INDEX

Function	SQL Statement
Rebuilding an index	REINDEX

Defining a Stored Procedure

A stored procedure is a set of SQL statements for achieving specific functions and is stored in the database after compiling. Users can specify a name and provide parameters (if necessary) to execute the stored procedure. For details about related SQL statements, see [Table 7-151](#).

Table 7-151 SQL statements for defining a stored procedure

Function	SQL Statement
Creating a stored procedure	CREATE PROCEDURE
Dropping a stored procedure	DROP PROCEDURE

Defining a Function

In GaussDB, a function is similar to a stored procedure, which is a set of SQL statements. The function and stored procedure are used the same. For details about related SQL statements, see [Table 7-152](#).

Table 7-152 SQL statements for defining a function

Function	SQL Statement
Creating a function	CREATE FUNCTION
Modifying attributes of a function or recompiling the function	ALTER FUNCTION
Dropping a function	DROP FUNCTION

Defining a View

A view is a virtual table exported from one or more basic tables. It is used to control data accesses of users. [Table 7-153](#) lists the related SQL statements.

Table 7-153 SQL statements for defining a view

Function	SQL Statement
Creating a view	CREATE VIEW
Dropping a view	DROP VIEW

Defining a Cursor

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers to context regions. With a cursor, the stored procedure can control alterations in context areas. For details, see [Table 7-154](#).

Table 7-154 SQL statements for defining a cursor

Function	SQL Statement
Creating a cursor	CURSOR
Moving a cursor	MOVE
Fetching data from a cursor	FETCH
Closing a cursor	CLOSE

Defining a Database Link

A database link can be used to remotely operate a database object. For details about corresponding SQL statements, see [Table 7-155](#).

Table 7-155 Database link-related SQL statements

Function	SQL Statement
Creating a database link	CREATE DATABASE LINK
Modifying a database link	ALTER DATABASE LINK
Deleting a database link	DROP DATABASE LINK

Defining a Foreign Data Wrapper

Table 7-156 SQL statements related to the foreign data wrapper

Function	SQL Statement
Creating a foreign data wrapper	CREATE FOREIGN DATA WRAPPER
Modifying a foreign data wrapper	ALTER FOREIGN DATA WRAPPER
Deleting a foreign data wrapper	DROP FOREIGN DATA WRAPPER

7.12 DML Syntax Overview

Data manipulation language (DML) is used to perform operations on data in database tables, such as inserting, updating, querying, or deleting data.

Inserting Data

Inserting data refers to adding one or multiple records to a database table. For details, see [INSERT](#).

Updating Data

Updating data refers to modifying one or multiple records in a database table. For details, see [UPDATE](#).

Querying Data

The database query statement **SELECT** is used to search required information in a database. For details, see [SELECT](#).

Deleting Data

GaussDB provides two statements for deleting data from database tables. To delete data meeting specified conditions from a database table, see [DELETE](#). To delete all data from a database table, see [TRUNCATE](#).

TRUNCATE can quickly delete all data from a database table, which achieves the effect same as that running **DELETE** to delete data without specifying conditions from each table. Deletion efficiency using **TRUNCATE** is faster because **TRUNCATE** does not scan tables. Therefore, **TRUNCATE** is useful in large tables.

Copying Data

GaussDB provides a statement for copying data between tables and files. For details, see [COPY](#).

Locking a Table

GaussDB provides multiple lock modes to control concurrent accesses to table data. For details, see [LOCK](#).

Calling a Function

GaussDB provides three statements for calling functions. These statements are the same in the syntax structure. For details, see [CALL](#).

Session Management

A session is a connection established between the user and the database. [Table 7-157](#) lists the related SQL statements.

Table 7-157 SQL statements related to sessions

Function	SQL Statement
Altering a session	ALTER SESSION
Killing a session	ALTER SYSTEM KILL SESSION

7.13 DCL Syntax Overview

Data control language (DCL) is used to create users and roles and set or modify database users or role rights.

Defining a Role

A role is used to manage permissions. For database security, management and operation permissions can be granted to different roles. For details about related SQL statements, see [Table 7-158](#).

Table 7-158 SQL statements for defining a role

Function	SQL Statement
Creating a role	CREATE ROLE
Altering role attributes	ALTER ROLE
Dropping a role	DROP ROLE

Defining a User

A user is used to log in to a database. Different permissions can be granted to users for managing data accesses and operations of the users. For details about related SQL statements, see [Table 7-159](#).

Table 7-159 SQL statements for defining a user

Function	SQL Statement
Creating a user	CREATE USER
Altering user attributes	ALTER USER
Dropping a user	DROP USER

Granting Rights

GaussDB provides a statement for granting rights to data objects and roles. For details, see [GRANT](#).

Revoking Rights

GaussDB provides a statement for revoking rights. For details, see [REVOKE](#).

Setting Default Rights

GaussDB allows users to set rights for objects that will be created in the future. For details, see [ALTER DEFAULT PRIVILEGES](#).

7.14 SQL Syntax

7.14.1 SQL Syntax

Table 7-160 SQL syntax

Format	Description
[]	The part enclosed in brackets ([]) is optional.
...	Preceding elements can appear repeatedly.
[x y ...]	One item is selected from two or more options or no item is selected.
{ x y ... }	One item is selected from two or more options.
[x y ...] [...]	Multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with spaces.
[x y ...] [,...]	Multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with commas (,).
{ x y ... } [...]	At least one parameter can be selected. If multiple parameters are selected, separate them with spaces.
{ x y ... } [,...]	At least one parameter can be selected. If multiple parameters are selected, separate them with commas (,).

7.14.2 ABORT

Description

Rolls back the current transaction and cancels the changes in the transaction.

It is equivalent to [ROLLBACK](#), and is present only for historical reasons. Now ROLLBACK is recommended.

Precautions

ABORT has no impact outside a transaction, but will provoke a warning.

Syntax

```
ABORT [ WORK | TRANSACTION ] ;
```


- The unified audit policy takes effect only after **enable_security_policy** is set to **on**.

Syntax

Add or delete an operation type in the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ]  
[ access_audit_clause ] };
```

Modify the filter criteria in the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause );
```

Delete the filter criteria from the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
```

Modify the description of the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;
```

Enable or disable the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ENABLE | DISABLE };
```

- **privilege_audit_clause**

DDL operation type and target resource label in the audit policy.

```
PRIVILEGES ( { DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ] )
```

- **access_audit_clause**

DML operation type and target resource label in the audit policy.

```
ACCESS ( { DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ] )
```

- **filter_group_clause**

Filter criteria in the audit policy.

```
FILTER ON { filter_type ( filter_value [, ... ] ) } [, ... ]
```

Parameters

- **policy_name**

Specifies the audit policy name, which must be unique.

Value range: a string. It must comply with the [naming convention](#).

- **DDL**

Specifies the operations that will be audited within the database: ALTER, ANALYZE, COMMENT, CREATE, DROP, GRANT, REVOKE, SET, and SHOW.

- **DML**

Specifies the operations that are audited within the database: COPY, DEALLOCATE, DELETE_P, EXECUTE, REINDEX, INSERT, PREPARE, SELECT, TRUNCATE, and UPDATE.

- **ALL**

Specifies all DDL or DML operations supported in the database. When the form is { DDL | ALL }, **ALL** indicates all DDL operations. When the form is { DML | ALL }, **ALL** indicates all DML operations.

- **filter_type**

Specifies a type of information to be filtered by the audit policy. The value can be **IP**, **ROLES**, or **APP**.

- **filter_value**
Indicates the detailed information to be filtered.
- **policy_comments**
Records description information of audit policies.
- **ENABLE|DISABLE**
Enables or disables the unified audit policy.

Examples

See [Examples](#) in section "CREATE AUDIT POLICY."

Helpful Links

[CREATE AUDIT POLICY](#) and [DROP AUDIT POLICY](#)

7.14.4 ALTER COLUMN ENCRYPTION KEY

Description

Encrypts the CMKs of CEKs in round robin (RR) mode and encrypts the plaintext of CEKs.

Precautions

- This syntax is specific to the fully-encrypted database. When connecting to the database server, enable the fully-encrypted database before using this syntax.
- This syntax takes effect on CMKs only. Encrypting the plaintext of CEKs does not change the ciphertext of the encrypted columns.

Syntax

```
ALTER COLUMN ENCRYPTION KEY column_encryption_key_name WITH VALUES ( CLIENT_MASTER_KEY = client_master_key_name );
```

Parameters

- **column_encryption_key_name**
Specifies the key name. In the same namespace, the value of this parameter must be unique.
Value range: a string. It must comply with the [naming convention](#).
- **client_master_key_name**
Specifies the CMK used to encrypt the CEK. The value is the CMK name, which is created using the CREATE CLIENT MASTER KEY syntax. The encrypted CMKs are different from those specified before RR encryption.

NOTICE

The constraints of using SM cryptographic algorithms are as follows:
SM2, SM3, and SM4 are SM cryptographic algorithms. To avoid legal risks, these algorithms must be used together. The SM cryptographic algorithms used for the RR encryption must be the same as those used before RR encryption.

Examples

See [7.15.58 - Examples](#).

Helpful Links

[CREATE COLUMN ENCRYPTION KEY](#) and [DROP COLUMN ENCRYPTION KEY](#)

7.14.5 ALTER COORDINATOR

Description

Changes the value of **nodeis_active** on a specified node in the **pgxc_node** system catalog. This operation can be performed on any normal CN in the cluster and also specifies the node on which the system catalog is to be modified.

Precautions

- ALTER COORDINATOR is a statement used to modify the system catalog. Only the administrator and users in internal maintenance mode (for example, CM) can execute this statement. This statement is dedicated for the CN removal feature and must be used together with other operations. You are advised not to run it by yourself.
- After this statement is executed, **select reload_active_coordinator()** needs to be called to update the connection pool information of the node on which the system catalog is modified.

Syntax

```
ALTER COORDINATOR nodename SET status  
WITH (nodename1[, nodename2, nodename3 ...]);
```

Parameters

- **nodename**
Specifies the node name corresponding to a row of records in the **pgxc_node** system catalog. After the node name is specified, the value of **nodeis_active** in the record is changed.
Value range: a string. Only CNs are supported. Ensure that the node name has a corresponding record in the **pgxc_node** system catalog.
- **status**
Specifies the updated value of **nodeis_active** in the **pgxc_node** system catalog.

Value range:

- FALSE
- TRUE

- **nodename1[, nodename2, nodename3 ...]**

Specifies the range of nodes on which the SQL statement is executed. When **ALTER COORDINATOR** is executed, the SQL statement is automatically delivered to all nodes in the range. The current execution node must be included.

Value range: a string. Only CNs are supported. Ensure that the node name has a corresponding record in the **pgxc_node** system catalog and the node state is normal. Otherwise, the SQL statement fails to be executed.

Examples

The cluster has three CNs, namely, cn_5001, cn_5002, and cn_5003, which are running properly.

If cn_5001 is faulty and needs to be removed from the cluster within the specified time, run the following SQL statement on cn_5002 and cn_5003 to change the value of **nodeis_active** corresponding to cn_5001 in the **pgxc_node** system catalog to **false**:

```
ALTER COORDINATOR cn_5001 SET False WITH (cn_5002,cn_5003);
```

After cn_5001 is recovered, run the following SQL statement on cn_5002 and cn_5003 to change the value of **nodeis_active** corresponding to cn_5001 in the **pgxc_node** system catalog to **true**:

```
ALTER COORDINATOR cn_5001 SET True WITH (cn_5002,cn_5003);
```

7.14.6 ALTER DATABASE

Description

Modifies a database, including its name, owner, connection limitation, and object isolation.

Precautions

- Only the database owner or a user granted with the ALTER permission can run the **ALTER DATABASE** command. System administrators have this permission by default. The following are permission constraints depending on attributes to be modified:
 - To modify the database name, you must have the CREATEDB permission.
 - To modify a database owner, you must be a database owner or system administrator and a member of the new owner role, with the CREATEDB permission.
 - To modify the default tablespace of the database, the user must have the CREATE permission on the tablespace. This statement physically migrates tables and indexes in a default tablespace to a new tablespace. Note that tables and indexes outside the default tablespace are not affected.
- You are not allowed to rename a database in use. To rename it, connect to another database.

Syntax

- Modify the maximum number of connections to the database.

```
ALTER DATABASE database_name  
[ WITH ] CONNECTION LIMIT connlimit;
```

- Rename the database.

```
ALTER DATABASE database_name  
RENAME TO new_name;
```

- Change the database owner.

```
ALTER DATABASE database_name  
OWNER TO new_owner;
```

- Change the default tablespace of the database.

```
ALTER DATABASE database_name  
SET TABLESPACE new_tablespace;
```

NOTE

If some tables or objects in the database have been created in `new_tablespace`, the default tablespace of the database cannot be changed to **new_tablespace**. An error will be reported during the execution.

- Modify the object isolation attribute of the database.

```
ALTER DATABASE database_name [ WITH ] { ENABLE | DISABLE } PRIVATE OBJECT;
```

NOTE

- To modify the object isolation attribute of a database, the database must be connected. Otherwise, the modification will fail.
- For a new database, the object isolation attribute is disabled by default. After the database object isolation attribute is enabled, the database automatically adds row-level security policies to the system catalogs `PG_CLASS`, `PG_ATTRIBUTE`, `PG_PROC`, `PG_NAMESPACE`, `PGXC_SLICE`, and `PG_PARTITION`. Common users can only view the objects (tables, functions, views, and columns) that they have the permission to access. This attribute does not take effect for administrators. After this attribute is enabled, administrators can still view all database objects.
- Change the database time zone.

```
ALTER DATABASE database_name SET DBTIMEZONE = time_zone;
```
- Modify the session parameter value of the database.

```
ALTER DATABASE database_name  
SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```
- Reset the database configuration parameter.

```
ALTER DATABASE database_name RESET  
{ configuration_parameter | ALL };
```

Parameters

- **database_name**
Specifies the name of the database whose attributes are to be modified.
Value range: a string. It must comply with the [naming convention](#).
- **connlimit**
Specifies the maximum number of concurrent connections that can be made to this database (excluding administrators' connections).
Value range: an integer ranging from -1 to $2^{31} - 1$. You are advised to set this parameter to an integer ranging from 1 to 50. The default value `-1` indicates that there is no restriction on the number of concurrent connections.
- **new_name**
Specifies the new name of a database.

Value range: a string. It must comply with the [naming convention](#).

- **new_owner**
Specifies the new owner of a database.
Value range: a string. It must be a valid username.
- **new_tablespace**
Specifies the new default tablespace of a database. The tablespace exists in the database. The default tablespace is **pg_default**.
Value range: a string. It must be a valid tablespace name.
- **configuration_parameter**
 - **value**
Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** closes the setting.

NOTICE

The current version does not support setting database-level parameters.

Value range: a string

- **DEFAULT**
- **OFF**
- **RESET**
- **FROM CURRENT**
Uses the value of **configuration_parameter** of the current session.
- **time_zone**
Sets the time zone of the database specified by **database_name**. You must have the permission on the corresponding database.
Value range: a string
 - Time zones supported by the system and their abbreviations
 - From - 15:59 to + 15:00
- **FROM CURRENT**
Sets the value of the database based on the current connected session.
- **RESET configuration_parameter**
Resets the specified database session parameter.

NOTICE

The current version does not support resetting database-level parameters.

- **RESET ALL**
Resets all database session parameters.

NOTICE

The current version does not support resetting database-level parameters.

NOTE

- Modify the default tablespace of a database by moving the table or index in the old tablespace into the new tablespace. This operation does not affect the tables or indexes in other non-default tablespaces.
- The modified database session parameter values will take effect in the next session.
- After setting the parameters, you need to manually run the **CLEAN CONNECTION** command to clear the old connections. Otherwise, the parameter values between cluster nodes may be inconsistent.

Examples

See [Examples](#) in section "CREATE DATABASE."

Helpful Links

[CREATE DATABASE](#) and [DROP DATABASE](#)

7.14.7 ALTER DATABASE LINK

Description

Modifies database link objects. For details about database links, see [DATABASE LINK](#).

Precautions

Currently, only the username and password can be modified for database links.

Syntax

```
ALTER [ PUBLIC ] DATABASE LINK dblink  
{ CONNECT TO user IDENTIFIED BY password };
```

Parameters

- **dblink**
Name of a connection.
- **user**
Username for connecting to a remote database.
- **password**
Password for connecting to a remote database.
- **PUBLIC**
Connection type. If **PUBLIC** is not specified, the database link is private by default.

Examples

```
-- Create an ORA-compatible data record.
gaussdb=# CREATE DATABASE ora_test_db DBCOMPATIBILITY 'ORA';

-- Switch to another database.
gaussdb=# \c ora_test_db

-- Create a user with the SYSADMIN permission.
ora_test_db=# CREATE USER user01 WITH SYSADMIN PASSWORD '*****';
ora_test_db=# SET ROLE user01 PASSWORD '*****';

-- Create a public database link.
ora_test_db=# CREATE PUBLIC DATABASE LINK public_dblink CONNECT TO 'user01' IDENTIFIED BY '*****'
USING (host '192.168.11.11',port '54399',dbname 'db01');

-- Create a common user.
ora_test_db=# CREATE USER user2 PASSWORD '*****';

-- Change database link object information.
ora_test_db=# ALTER PUBLIC DATABASE LINK public_dblink CONNECT TO 'user2' IDENTIFIED BY '*****';

-- Delete a public database link.
ora_test_db=# DROP PUBLIC DATABASE LINK public_dblink;

-- Delete the user.
ora_test_db=# RESET ROLE;
ora_test_db=# DROP USER user01;
ora_test_db=# DROP USER user2;

-- Switch back to the initial database and delete the test database. Replace postgres with the actual
database name.
ora_test_db=# \c postgres
gaussdb=# DROP DATABASE ora_test_db;
```

Helpful Links

[CREATE DATABASE LINK](#) and [DROP DATABASE LINK](#)

7.14.8 ALTER DEFAULT PRIVILEGES

Function

Allows you to set the permissions that will be applied to objects created in the future. (It does not affect permissions granted to existing objects.)

Precautions

Currently, you can change only the permissions for tables (including views), sequences, functions, client master keys of encrypted databases, column encryption keys, and types.

Syntax

```
ALTER DEFAULT PRIVILEGES
  [ FOR { ROLE | USER } target_role [, ...] ]
  [ IN SCHEMA schema_name [, ...] ]
  abbreviated_grant_or_revoke;
```

- **abbreviated_grant_or_revoke** grants or revokes permissions on some objects.
grant_on_tables_clause
| grant_on_sequences_clause
| grant_on_functions_clause
| grant_on_types_clause

```
| grant_on_client_master_keys_clause
| grant_on_column_encryption_keys_clause
| revoke_on_tables_clause
| revoke_on_sequences_clause
| revoke_on_functions_clause
| revoke_on_types_clause
| revoke_on_client_master_keys_clause
| revoke_on_column_encryption_keys_clause
```

- **grant_on_tables_clause** grants permissions on tables.

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP |
COMMENT | INDEX | VACUUM }
[, ...] | ALL [ PRIVILEGES ] }
ON TABLES
TO { [ GROUP ] role_name | PUBLIC } [, ...]
[ WITH GRANT OPTION ]
```
- **grant_on_sequences_clause** grants permissions on sequences.

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
[, ...] | ALL [ PRIVILEGES ] }
ON SEQUENCES
TO { [ GROUP ] role_name | PUBLIC } [, ...]
[ WITH GRANT OPTION ]
```
- **grant_on_functions_clause** grants permissions on functions.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
ON FUNCTIONS
TO { [ GROUP ] role_name | PUBLIC } [, ...]
[ WITH GRANT OPTION ]
```
- **grant_on_types_clause** grants permissions on types.

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
ON TYPES
TO { [ GROUP ] role_name | PUBLIC } [, ...]
[ WITH GRANT OPTION ]
```
- **grant_on_client_master_keys_clause** grants permissions on CMKs.

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
ON CLIENT_MASTER_KEYS
TO { [ GROUP ] role_name | PUBLIC } [, ...]
[ WITH GRANT OPTION ]
```
- **grant_on_column_encryption_keys_clause** grants permissions on column encryption keys.

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
ON COLUMN_ENCRYPTION_KEYS
TO { [ GROUP ] role_name | PUBLIC } [, ...]
[ WITH GRANT OPTION ]
```
- **revoke_on_tables_clause** revokes permissions on tables.

```
REVOKE [ GRANT OPTION FOR ]
{ { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }
[, ...] | ALL [ PRIVILEGES ] }
ON TABLES
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```
- **revoke_on_sequences_clause** revokes permissions on sequences.

```
REVOKE [ GRANT OPTION FOR ]
{ { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
[, ...] | ALL [ PRIVILEGES ] }
ON SEQUENCES
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```
- **revoke_on_functions_clause** revokes permissions on functions.

```
REVOKE [ GRANT OPTION FOR ]
{ { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
ON FUNCTIONS
FROM { [ GROUP ] role_name | PUBLIC } [, ...]
[ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- **revoke_on_types_clause** revokes permissions on types.
REVOKE [GRANT OPTION FOR]
{ { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON TYPES
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT | CASCADE CONSTRAINTS]
- **revoke_on_client_master_keys_clause** revokes permissions on CMKs.
REVOKE [GRANT OPTION FOR]
{ { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
ON CLIENT_MASTER_KEYS
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT | CASCADE CONSTRAINTS]
- **revoke_on_column_encryption_keys_clause** revokes permissions on CEKs.
REVOKE [GRANT OPTION FOR]
{ { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
ON COLUMN_ENCRYPTION_KEYS
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT | CASCADE CONSTRAINTS]

Parameter Description

- **target_role**
Specifies the name of an existing role. If **FOR ROLE/USER** is omitted, the current role is assumed.
Value range: an existing role name
- **schema_name**
Specifies the name of an existing schema.
target_role must have the **CREATE** permission for **schema_name**.
Value range: an existing schema name
- **role_name**
Specifies the name of an existing role to grant or revoke permissions for.
Value range: an existing role name

NOTICE

To drop a role for which the default permissions have been granted, reverse the changes in its default permissions or use **DROP OWNED BY** to get rid of the default permission entry for the role.

Examples

```
-- Grant the SELECT permission on all the tables (and views) in tpcds to every user.
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLES TO PUBLIC;

-- Create a common user jack.
gaussdb=# CREATE USER jack PASSWORD '*****';

-- Grant the INSERT permission on all the tables in tpcds to the user jack.
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLES TO jack;

-- Revoke the preceding permissions.
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABLES FROM PUBLIC;
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABLES FROM jack;

-- Delete user jack.
gaussdb=# DROP USER jack;
```

Helpful Links

[GRANT](#) and [REVOKE](#)

7.14.9 ALTER DIRECTORY

Description

Modifies directory attributes.

Precautions

- Currently, only the directory owner can be changed.
- When **enable_access_server_directory** is set to **off**, only the initial user is allowed to change the directory owner. When **enable_access_server_directory** is set to **on**, users with the SYSADMIN permission and the directory object owner can change the directory object owner, and the user who changes the owner is required to be a member of the new owner.

Syntax

```
ALTER DIRECTORY directory_name  
OWNER TO new_owner;
```

Parameters

- **directory_name**
Specifies the name of a directory to be modified. The value must be an existing directory name.
- **new_owner**
Specifies the new owner of the directory.

Examples

```
-- Create a directory.  
gaussdb=# CREATE OR REPLACE DIRECTORY dir as '/tmp/';  
  
-- Create a user.  
gaussdb=# CREATE USER jim PASSWORD '*****';  
  
-- Change the owner of the directory.  
gaussdb=# ALTER DIRECTORY dir OWNER TO jim;  
  
-- Delete the directory.  
gaussdb=# DROP DIRECTORY dir;  
  
-- Delete the user.  
gaussdb=# DROP USER jim;
```

Helpful Links

[CREATE DIRECTORY](#) and [DROP DIRECTORY](#)

7.14.10 ALTER FOREIGN DATA WRAPPER

Description

Modifies the definition of a foreign data wrapper.

Precautions

- Only initial users and system administrators can modify the foreign data wrapper.
- The alter statement can be successfully executed only when **support_extended_features** is set to **on**.

Syntax

- Set the attributes of the foreign data wrapper.

```
ALTER FOREIGN DATA WRAPPER name  
[ HANDLER handler_function | NO HANDLER ]  
[ VALIDATOR validator_function | NO VALIDATOR ]  
[ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] ) ];
```
- Set a new owner.

```
ALTER FOREIGN DATA WRAPPER name OWNER TO new_owner;
```
- Set a new name.

```
ALTER FOREIGN DATA WRAPPER name RENAME TO new_name;
```

Parameters

- **name**
Specifies a name of an existing foreign data wrapper.
- **HANDLER handler_function**
Specifies a new handler function for a foreign data wrapper.
- **NO HANDLER**
Specifies that the foreign data wrapper no longer has the handler function.

 **CAUTION**

Foreign tables that use foreign data wrapper but do not have handlers cannot be accessed.

- **VALIDATOR validator_function**
Specifies a new validator function for a foreign data wrapper.

 **CAUTION**

Depending on the new validator, an existing option for the foreign data wrapper or dependent server, user mapping, or foreign table may be invalid. Before using a foreign data wrapper, ensure that these options are correct. However, any options specified in the **ALTER FOREIGN DATA WRAPPER** command will be checked using the new validator function.

- **NO VALIDATOR**
Specifies that the foreign data wrapper no longer has the validator function.
- **OPTIONS ([ADD | SET | DROP] option ['value'] [, ...])**
Modifies the options of a foreign data wrapper. ADD, SET, and DROP specify the actions to be performed. If no operation is specified, the default operation is ADD. The option name must be unique. When foreign data is used to wrap validator functions, names and values are also validated.
- **new_owner**
Specifies the username of the new owner for a foreign data wrapper.
- **new_name**
Specifies the new name of a foreign data wrapper.

Example

```
-- Modify a foreign data wrapper dbi, add the foo option, and delete bar.
gaussdb=# ALTER FOREIGN DATA WRAPPER dbi OPTIONS (ADD foo '1', DROP 'bar');

-- Change the validator of the foreign data wrapper dbi to bob.myvalidator.
gaussdb=# ALTER FOREIGN DATA WRAPPER dbi VALIDATOR bob.myvalidator;
```

Helpful Links

[CREATE FOREIGN DATA WRAPPER](#) and [DROP FOREIGN DATA WRAPPER](#)

7.14.11 ALTER FUNCTION

Description

Modifies the attributes of a user-defined function or recompiles a function.

Precautions

- Only the function owner or a user granted with the ALTER permission can run the **ALTER FUNCTION** command. A system administrator has this permission by default. The following are permission constraints depending on attributes to be modified:
 - If a function involves operations on temporary tables, ALTER FUNCTION cannot be used.
 - To modify the owner or schema of a function, you must be a function owner or system administrator and a member of the new owner role.
 - Only system administrators and the initial user can change the schema of a function to public.
- The **plpgsql_dependency** parameter must be set for function recompilation.
- Only the initial user or the user who creates the stored procedure can change the stored procedure to a stored procedure that has the definer permission.
- When separation of duties is enabled, no role is allowed to modify the owner of a function with the definer permission.
- When separation of duties is disabled, only the initial user and system administrator can change the owner of a function with the definer

permission. However, the function owner cannot be changed to an O&M administrator.

- Only the initial user can change the owner of a function to the initial user.

Syntax

- Modify the additional parameters of the user-defined function.

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )  
action [ ... ] [ RESTRICT ];
```

The syntax of the **action** clause is as follows:

```
{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}  
| {IMMUTABLE | STABLE | VOLATILE}  
| {SHIPPABLE | NOT SHIPPABLE}  
| {NOT FENCED | FENCED}  
| [ NOT ] LEAKPROOF  
| { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }  
| AUTHID { DEFINER | CURRENT_USER }  
| COST execution_cost  
| ROWS result_rows  
| SET configuration_parameter { { TO | = } { value | DEFAULT } } FROM CURRENT}  
| RESET {configuration_parameter | ALL}
```

- Rename the user-defined function.

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )  
RENAME TO new_name;
```
- Change the owner of the user-defined function.

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )  
OWNER TO new_owner;
```
- Change the schema of the user-defined function.

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )  
SET SCHEMA new_schema;
```
- Recompile the function.

```
ALTER FUNCTION function_name COMPILE;
```

Parameters

- **function_name**
Specifies the name of the function to be modified.
Value range: an existing function name
- **argmode**
Specifies whether a parameter is an input or output parameter.
Value range:
 - **IN**: declares input parameters.
 - **OUT**: declares output parameters.
 - **INOUT**: declares input and output parameters.
- **argname**
Parameter name.
Value range: a string. It must comply with the [naming convention](#).
- **argtype**
Specifies the parameter type.
Value range: a valid type. For details, see [Data Type](#).
- **CALLED ON NULL INPUT**

Declares that some parameters of the function can be called in normal mode if the parameter values are null. Omitting this parameter is the same as specifying it.

- **RETURNS NULL ON NULL INPUT**

STRICT

Specifies that the function always returns **NULL** whenever any of its parameters is **NULL**. If this parameter is specified, the function is not executed when there is **NULL** parameter; instead a **NULL** result is assumed automatically.

RETURNS NULL ON NULL INPUT and **STRICT** have the same functions.

- **IMMUTABLE**

Specifies that the function always returns the same result if the parameter values are the same.

- **STABLE**

Specifies that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.

- **VOLATILE**

Specifies that the function value can change in a single table scan and no optimization is performed.

- **SHIPPABLE**

- **NOT SHIPPABLE**

Specifies whether the function can be pushed down to DN for execution.

Functions of the **IMMUTABLE** type can always be pushed down to DN.

Functions of the **STABLE** or **VOLATILE** type can be pushed down to DN only if their attribute is **SHIPPABLE**.

- **LEAKPROOF**

Specifies that the function has no side effect and the parameter contains only the return value. **LEAKPROOF** can be set only by system administrators.

- **EXTERNAL**

(Optional) The purpose is to be compatible with SQL. This feature applies to all functions, not only external functions.

- **SECURITY INVOKER**

AUTHID CURREN_USER

Specifies that the function will be executed with the permissions of the user who calls it. Omitting this parameter is the same as specifying it.

SECURITY INVOKER and **AUTHID CURREN_USER** have the same functions.

- **SECURITY DEFINER**

AUTHID DEFINER

Specifies that the function will be executed with the permissions of the user who created it.

AUTHID DEFINER and **SECURITY DEFINER** have the same functions.

- **COST execution_cost**

Estimates the execution cost of a function.

The unit of **execution_cost** is **cpu_operator_cost**.

Value range: a positive integer

- **ROWS result_rows**

Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.

Value range: a positive number. The default value is **1000**.

- **configuration_parameter**

- **value**

Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** closes the setting.

Value range: a string

- **DEFAULT**

- **OFF**

- **RESET**

Specifies the default value.

- **from current**

Uses the value of **configuration_parameter** of the current session.

- **new_name**

Specifies the new name of a function. To change the schema of a function, you must have the CREATE permission on the new schema.

Value range: a string. It must comply with the [naming convention](#).

- **new_owner**

Specifies the new owner of a function. To change the owner of a function, the new owner must have the CREATE permission on the schema to which the function belongs. Note that only the initial user can set the function owner to another initial user.

Value range: an existing user role

- **new_schema**

Specifies the new schema of a function.

Value range: an existing schema

Examples

See [Examples](#) in section "CREATE FUNCTION."

Recompilation examples:

```
-- Enable the dependency function.
gaussdb=# set behavior_compat_options='plpgsql_dependency';
-- Create a function.
gaussdb=# create or replace function test_func(a int) return int
is
    proc_var int;
begin
    proc_var := a;
    return 1;
end;
```

```
/
-- Recompile the function with a function name.
gaussdb=# alter procedure test_func compile;
-- Recompile the stored procedure with a signed int type function.
gaussdb=# alter procedure test_func(int) compile;
```

Helpful Links

[CREATE FUNCTION](#) and [DROP FUNCTION](#)

7.14.12 ALTER GLOBAL CONFIGURATION

Description

Adds or modifies the key-value of the `gs_global_config` system catalog. You can modify an existing parameter or add a new one.

Precautions

- Only the initial database user can run this command.
- The parameter name cannot be **weak_password** or **undostoragetype**.

Syntax

```
ALTER GLOBAL CONFIGURATION with(name=value, name=value...);
```

Parameters

- **name**
Parameter name, which is of the text type. The value cannot be **weak_password** or **undostoragetype**.
- **value**
Parameter value, which is of the text type.

Examples

```
-- Insert.
gaussdb=# ALTER GLOBAL CONFIGURATION with(redis_is_ok = true);
-- Query.
gaussdb=# SELECT * FROM gs_global_config;
   name   | value
-----+-----
 buckets_len | 16384
 undostoragetype | page
 redis_is_ok  | true
(3 rows)

-- Modify.
gaussdb=# ALTER GLOBAL CONFIGURATION with(redis_is_ok = false);
-- Query.
gaussdb=# SELECT * FROM gs_global_config;
   name   | value
-----+-----
 buckets_len | 16384
 undostoragetype | page
 redis_is_ok  | false
(3 rows)

-- Delete.
gaussdb=# DROP GLOBAL CONFIGURATION redis_is_ok;
```

```
-- Query.
gaussdb=# SELECT * FROM gs_global_config;
   name   | value
-----+-----
 buckets_len | 16384
 undostorage_type | page
(2 rows)
```

Helpful Links

[DROP GLOBAL CONFIGURATION](#)

7.14.13 ALTER GROUP

Function

ALTER GROUP modifies the attributes of a user group.

Precautions

ALTER GROUP is an alias for **ALTER ROLE**, and it is not a standard SQL syntax and not recommended. Users can use **ALTER ROLE** directly.

Syntax

- Add users to a group.
ALTER GROUP group_name
ADD USER user_name [, ...];
- Remove users from a group.
ALTER GROUP group_name
DROP USER user_name [, ...];
- Change the name of the group.
ALTER GROUP group_name
RENAME TO new_name;

Parameter Description

See [Parameters](#) in section "ALTER ROLE."

Examples

```
-- Create a user group.
gaussdb=# CREATE GROUP super_users WITH PASSWORD "*****";

-- Create a user.
gaussdb=# CREATE ROLE lche WITH PASSWORD "*****";

-- Create a user.
gaussdb=# CREATE ROLE jim WITH PASSWORD "*****";

-- Add users to a user group.
gaussdb=# ALTER GROUP super_users ADD USER lche, jim;

-- Remove users from a user group.
gaussdb=# ALTER GROUP super_users DROP USER jim;

-- Change the name of a user group.
gaussdb=# ALTER GROUP super_users RENAME TO normal_users;

-- Delete the user.
```

```
gaussdb=# DROP ROLE lche, jim;
-- Delete the user group.
gaussdb=# DROP GROUP normal_users;
```

Helpful Links

[CREATE GROUP](#), [DROP GROUP](#), and [ALTER ROLE](#)

7.14.14 ALTER INDEX

Description

ALTER INDEX modifies the definition of an existing index.

It has the following forms:

- IF EXISTS
Sends a notice instead of an error if the specified index does not exist.
- RENAME TO
Changes only the name of the index. The stored data is not affected.
- SET TABLESPACE
Changes the index tablespace to the specified tablespace and moves index-related data files to the new tablespace.
- SET ({ STORAGE_PARAMETER = value } [, ...])
Changes one or more index-method-specific storage parameters of an index. Note that the index content will not be modified immediately by this statement. You may need to use REINDEX to rebuild the index based on different parameters to achieve the expected effect.
- RESET ({ storage_parameter } [, ...])
Resets one or more index-method-specific storage parameters of an index to the default value. Similar to the SET statement, REINDEX may be used to completely update the index.
- [MODIFY PARTITION index_partition_name] UNUSABLE
Sets the indexes on a table or index partition to be unavailable.
- REBUILD [PARTITION index_partition_name]
Rebuilds indexes on a table or an index partition.
- RENAME PARTITION
Renames an index partition.
- MOVE PARTITION
Modifies the tablespace to which an index partition belongs.

Precautions

Only the index owner, a user who has the INDEX permission on the table where the index resides, or a user who has the ALTER ANY INDEX permission can run the **ALTER INDEX** command. The system administrator has this permission by default.

Syntax

- Rename a table index.

```
ALTER INDEX [ IF EXISTS ] index_name  
  RENAME TO new_name;
```
- Change the tablespace to which a table index belongs.

```
ALTER INDEX [ IF EXISTS ] index_name  
  SET TABLESPACE tablespace_name;
```
- Modify the storage parameter of a table index.

```
ALTER INDEX [ IF EXISTS ] index_name  
  SET ( {storage_parameter = value} [, ... ] );
```
- Reset the storage parameter of a table index.

```
ALTER INDEX [ IF EXISTS ] index_name  
  RESET ( storage_parameter [, ... ] );
```
- Set a table index or an index partition to be unavailable.

```
ALTER INDEX [ IF EXISTS ] index_name  
  [ MODIFY PARTITION index_partition_name ] UNUSABLE;
```
- Rebuild a table index or index partition.

```
ALTER INDEX index_name  
  REBUILD [ PARTITION index_partition_name ];
```
- Rename an index partition.

```
ALTER INDEX [ IF EXISTS ] index_name  
  RENAME PARTITION index_partition_name TO new_index_partition_name;
```
- Modify the tablespace to which an index partition belongs.

```
ALTER INDEX [ IF EXISTS ] index_name  
  MOVE PARTITION index_partition_name TABLESPACE new_tablespace;
```

Parameters

- **index_name**
Specifies the index name to be modified.
- **new_name**
Specifies the new name of the index.
Value range: a string. It must comply with the [naming convention](#).
- **tablespace_name**
Specifies the tablespace name.
Value range: an existing tablespace name.
- **storage_parameter**
Specifies the name of an index-method-specific parameter. **ACTIVE_PAGES** indicates the number of index pages, which may be less than the actual number of physical file pages and can be used for optimization. Currently, this parameter is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM and ANALYZE (including AUTOVACUUM). You are not advised to manually set this parameter because it is invalid in distributed mode.
- **value**
Specifies the new value for an index-method-specific storage parameter. This might be a number or a word depending on the parameter.
- **new_index_partition_name**
Specifies the new name of the index partition.
- **index_partition_name**

Specifies the name of an index partition.

- **new_tablespace**
Specifies a new tablespace.

Examples

See [Examples](#) in section "CREATE INDEX."

Helpful Links

[CREATE INDEX](#), [DROP INDEX](#), and [REINDEX](#)

7.14.15 ALTER LANGUAGE

This version does not support this syntax.

7.14.16 ALTER MASKING POLICY

Description

Modifies a masking policy.

Precautions

- Only POLADMIN, SYSADMIN, or the initial user can perform this operation.
- The masking policy takes effect only after **enable_security_policy** is set to **on**.

Syntax

- Modify the policy description.
`ALTER MASKING POLICY policy_name COMMENTS policy_comments;`
- Modify the masking method.
`ALTER MASKING POLICY policy_name [ADD | REMOVE | MODIFY] masking_actions[, ...];`
The syntax of **masking_action**.
`masking_function ON LABEL(label_name[, ...])`
- Modify the scenarios where the masking policies take effect.
`ALTER MASKING POLICY policy_name MODIFY(FILTER ON FILTER_TYPE(filter_value[, ...])[, ...]);`
- Remove the filters of the masking policies.
`ALTER MASKING POLICY policy_name DROP FILTER;`
- Enable or disable the masking policies.
`ALTER MASKING POLICY policy_name [ENABLE | DISABLE];`

Parameters

- **policy_name**
Specifies the masking policy name, which must be unique.
Value range: a string. It must comply with the [naming convention](#).
- **policy_comments**
Adds or modifies description of masking policies.
- **masking_function**
Specifies eight preset masking methods or user-defined functions. Schema is supported.

maskall is not a preset function. It is hard-coded and cannot be displayed by running `\df`.

The preset masking methods are as follows:

```
maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |  
shufflemasking | alldigitsmasking | regepmasking
```

- **label_name**
Specifies the resource label name.
- **FILTER_TYPE**
Specifies the types of information to be filtered by the policies: **IP**, **ROLES**, and **APP**.
- **filter_value**
Indicates the detailed information to be filtered, such as the IP address, app name, and username.
- **ENABLE|DISABLE**
Enables or disables the masking policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

```
-- Create users dev_mask and bob_mask.  
gaussdb=# CREATE USER dev_mask PASSWORD '*****';  
gaussdb=# CREATE USER bob_mask PASSWORD '*****';  
  
-- Create table tb_for_masking.  
gaussdb=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);  
  
-- Create a resource label for sensitive column col1.  
gaussdb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);  
  
-- Create a resource label for sensitive column col2.  
gaussdb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);  
  
-- Create a masking policy for the operation of accessing sensitive column col1.  
gaussdb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);  
  
-- Add description for masking policy maskpol1.  
gaussdb=# ALTER MASKING POLICY maskpol1 COMMENTS 'masking policy for tb_for_masking.col1';  
  
-- Modify masking policy maskpol1 to add a masking method.  
gaussdb=# ALTER MASKING POLICY maskpol1 ADD randommasking ON LABEL(mask_lb2);  
  
-- Modify masking policy maskpol1 to remove a masking method.  
gaussdb=# ALTER MASKING POLICY maskpol1 REMOVE randommasking ON LABEL(mask_lb2);  
  
-- Modify masking policy maskpol1 to modify a masking method.  
gaussdb=# ALTER MASKING POLICY maskpol1 MODIFY randommasking ON LABEL(mask_lb1);  
  
-- Modify masking policy maskpol1 so that it takes effect only for scenarios where users are dev_mask and bob_mask, the client tool is gsq, and the IP addresses are 10.20.30.40 and 127.0.0.24.  
gaussdb=# ALTER MASKING POLICY maskpol1 MODIFY (FILTER ON ROLES(dev_mask, bob_mask), APP(gsql), IP('10.20.30.40', '127.0.0.24'));  
  
-- Modify masking policy maskpol1 so that it takes effect for all user scenarios.  
gaussdb=# ALTER MASKING POLICY maskpol1 DROP FILTER;  
  
-- Disable masking policy maskpol1.  
gaussdb=# ALTER MASKING POLICY maskpol1 DISABLE;  
  
-- Delete a masking policy.  
gaussdb=# DROP MASKING POLICY maskpol1;
```

```
-- Delete resource labels.  
gaussdb=# DROP RESOURCE LABEL mask_lb1, mask_lb2;  
  
-- Delete the tb_for_masking table.  
gaussdb=# DROP TABLE tb_for_masking;  
  
-- Delete the dev_mask and bob_mask users.  
gaussdb=# DROP USER dev_mask, bob_mask;
```

Helpful Links

[CREATE MASKING POLICY](#) and [DROP MASKING POLICY](#)

7.14.17 ALTER MATERIALIZED VIEW

Function

ALTER MATERIALIZED VIEW changes multiple auxiliary attributes of an existing materialized view.

Statements and actions that can be used for **ALTER MATERIALIZED VIEW** are a subset of **ALTER TABLE** and have the same meaning when used for materialized views. For details, see [ALTER TABLE](#).

Precautions

- Only the owner of a materialized view or a system administrator has the **ALTER TMATERIALIZED VIEW** permission.
- The materialized view structure cannot be modified.

Syntax

- Change the owner of a materialized view.

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name  
OWNER TO new_owner;
```
- Modify the column of a materialized view.

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name  
RENAME [ COLUMN ] column_name TO new_column_name;
```
- Rename a materialized view.

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name  
RENAME TO new_name;
```

Parameter Description

- **mv_name**
Specifies the name of an existing materialized view, which can be schema-qualified.
Value range: a string. It must comply with the [naming convention](#).
- **column_name**
Specifies the name of a new or existing column.
Value range: a string. It must comply with the [naming convention](#).
- **new_column_name**
Specifies the new name of an existing column.

- **new_owner**
Specifies the username of the new owner of a materialized view.
- **new_name**
Specifies the new name of a materialized view.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int) WITH(STORAGE_TYPE=ASTORE);

-- Create a complete-refresh materialized view.
gaussdb=# CREATE MATERIALIZED VIEW foo AS SELECT * FROM my_table;

-- Rename the materialized view foo to bar.
gaussdb=# ALTER MATERIALIZED VIEW foo RENAME TO bar;

-- Delete a complete-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW bar;

-- Delete the my_table table.
gaussdb=# DROP TABLE my_table;
```

Helpful Links

[CREATE INCREMENTAL MATERIALIZED VIEW](#), [CREATE MATERIALIZED VIEW](#), [DROP MATERIALIZED VIEW](#), [REFRESH INCREMENTAL MATERIALIZED VIEW](#), and [REFRESH MATERIALIZED VIEW](#)

7.14.18 ALTER NODE

Description

Modifies the definition of an existing node.

Precautions

ALTER NODE is an API of the cluster management tool and is used to manage clusters. Only administrators have the permission to use this API. You are advised not to use this API, because doing so affects the cluster.

Syntax

```
ALTER NODE nodename WITH
(
  [ TYPE = nodetype,]
  [ HOST = hostname,]
  [ PORT = portnum,]
  [ HOST1 = 'hostname',]
  [ PORT1 = portnum,]
  [ HOSTPRIMARY [ = boolean ],]
  [ PRIMARY [ = boolean ],]
  [ PREFERRED [ = boolean ],]
  [ SCTP_PORT = portnum,]
  [ CONTROL_PORT = portnum,]
  [ SCTP_PORT1 = portnum,]
  [ CONTROL_PORT1 = portnum, ]
  [ NODEIS_CENTRAL [ = boolean ]]
);
```

 NOTE

The port whose number is specified by **PORT** is used for internal communications between nodes. Unlike the port connecting to an external client, it can be queried in the **pgxc_node** table.

Parameters

See [Parameters](#) in "CREATE NODE."

Examples

```
-- Create cluster nodes on all CNs.
gaussdb=# CREATE NODE datanode1 WITH(
  TYPE = datanode,
  PREFERRED = false
);

-- Query the initial cluster DN status.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----+-----
 datanode1 | f
 datanode2 | f
(2 rows)

-- Set datanode1 as the preferred DN.
gaussdb=# ALTER NODE datanode1 WITH(preferred = true);

-- Query the cluster DN status after the change.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----+-----
 datanode1 | t
 datanode2 | f
(2 rows)

-- Connect to only one CN to delete cluster nodes.
gaussdb=# DROP NODE datanode1;
```

Helpful Links

[CREATE NODE](#) and [DROP NODE](#)

7.14.19 ALTER NODE GROUP

Description

Modifies information about a node group.

Precautions

- Only a system administrator or a user who has the ALTER permission of a node group can modify the information about the node group.
- Node group modification is an internal operation of the system. Except SET DEFAULT, other operations must be performed in maintenance mode (by calling **set xc_maintenance_mode=on;**).
- ALTER NODE GROUP can be used only within a database. To avoid data inconsistency in DBMS, do not manually run this SQL statement.

Syntax

```
ALTER NODE GROUP groupname
{
| SET DEFAULT
| RENAME TO new_group_name
| SET VCGROUP RENAME TO new_group_name
| SET NOT VCGROUP
| SET TABLE GROUP new_group_name
| COPY BUCKETS FROM src_group_name
| ADD NODE ( nodename [, ... ] )
| DELETE NODE ( nodename [, ... ] )
| RESIZE TO dest_group_name
| SET VCGROUP WITH GROUP new_group_name
};
```

Parameters

- **groupname**
Specifies the name of the node group to be modified.
Value range: a string. It must comply with the [naming convention](#).
- **SET DEFAULT**
Sets **in_redistribution** to 'y' for all node groups excluding the one specified by **groupname**. To be compatible with earlier versions, this syntax is retained and does not need to be executed in maintenance mode.
- **RENAME TO new_group_name**
Renames the node group specified by **groupname** to **new_group_name**.
- **SET TABLE GROUP new_group_name**
Changes all the **group_names** in the **pggroup** columns of the **pgxc_class** tables on all CNs to **new_group_name**.
- **COPY BUCKETS FROM src_group_name**
Copies values in the **group_members** and **group_buckets** columns from the node group specified by **src_group_name** to the node group specified by **groupname**.
- **ADD NODE (nodename [, ...])**
Adds nodes from the node group specified by **groupname**. After the statement execution, the new nodes are registered with the **PGXC_NODE** system catalog. This statement only modifies the system catalog and does not add nodes or redistribute data. Do not call this statement. You can observe the impact of the statement in the **PGXC_GROUP** system catalog.
- **DELETE NODE (nodename [, ...])**
Deletes nodes from the node group specified by **groupname**. The deleted nodes still exist in the **PGXC_NODE** system catalog. This statement only modifies the system catalog and does not delete nodes or redistribute data. Do not call this statement. You can observe the impact of the statement in the **PGXC_GROUP** system catalog.
- **RESIZE TO dest_group_name**
Specifies a resize flag for the cluster. Set **groupname** to the source node group before data redistribution and cancel the **is_installation** flag. Set **dest_group_name** to the destination node group and set the **is_installation** flag.

Examples

```
-- Query the initial cluster DN status.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
node_name | nodeis_preferred
-----+-----
dn_6001_6002_6003 | f
dn_6004_6005_6006 | f
dn_6007_6008_6009 | f
(3 rows)

-- Create a node group and replace dn_6001_6002_6003 with the actual node name obtained in the
previous step.
gaussdb=# CREATE NODE GROUP test_group WITH ( dn_6001_6002_6003 );

-- Change the name of the created node group to test_group_new. This must be implemented in the
maintenance mode (by setting xc_maintenance_mode to on);
gaussdb=# SET xc_maintenance_mode=on;
gaussdb=# ALTER NODE GROUP test_group RENAME TO test_group_new;

-- Query the node group.
gaussdb=# SELECT group_name, group_members FROM pgxc_group;
group_name | group_members
-----+-----
group_version1 | 16384 16388 16394
test_group_new | 16384
(2 rows)

-- Delete the node group and disable the maintenance mode.
gaussdb=# DROP NODE GROUP test_group_new;
gaussdb=# SET xc_maintenance_mode=off;
```

Helpful Links

[CREATE NODE GROUP](#) and [DROP NODE GROUP](#)

7.14.20 ALTER RESOURCE LABEL

Description

Modifies resource labels.

Precautions

Only users with the poladmin or sysadmin permission, or the initial user can perform this operation.

Syntax

```
ALTER RESOURCE LABEL label_name (ADD|REMOVE)
label_item_list[, ...];
```

- **label_item_list**
resource_type(resource_path[, ...])
- **resource_type**
TABLE | COLUMN | SCHEMA | VIEW | FUNCTION

Parameters

- **label_name**
Specifies the resource label name.
Value range: a string. It must comply with the [naming convention](#).

- **resource_type**
Specifies the type of database resources to be labeled.
- **resource_path**
Specifies the path of database resources.

Examples

```
-- Create basic table table_for_label.
gaussdb=# CREATE TABLE table_for_label(col1 int, col2 text);

-- Create resource label table_label.
gaussdb=# CREATE RESOURCE LABEL table_label ADD COLUMN(table_for_label.col1);

-- Attach resource label table_label to col2.
gaussdb=# ALTER RESOURCE LABEL table_label ADD COLUMN(table_for_label.col2)

-- Remove table_label from an item.
gaussdb=# ALTER RESOURCE LABEL table_label REMOVE COLUMN(table_for_label.col1);

-- Delete the resource label table_label.
gaussdb=# DROP RESOURCE LABEL table_label;

-- Delete the base table table_for_label.
gaussdb=# DROP TABLE table_for_label;
```

Helpful Links

[CREATE RESOURCE LABEL](#) and [DROP RESOURCE LABEL](#)

7.14.21 ALTER ROLE

Description

Modifies role attributes.

Precautions

None

Syntax

- Modify the permissions of a role.
ALTER ROLE role_name [[WITH] option [...]];

The option clause for granting permissions is as follows:

```
{CREATEDB | NOCREATEDB}
| {CREATEROLE | NOCREATEROLE}
| {INHERIT | NOINHERIT}
| {AUDITADMIN | NOAUDITADMIN}
| {SYSADMIN | NOSYSADMIN}
| {MONADMIN | NOMONADMIN}
| {OPRADMIN | NOOPRADMIN}
| {POLADMIN | NOPOLADMIN}
| {USEFT | NOUSEFT}
| {LOGIN | NOLOGIN}
| {REPLICATION | NOREPLICATION}
| {VCADMIN | NOVADMIN}
| {PERSISTENCE | NOPERSISTENCE}
| CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'[EXPIRED]
| [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY 'password' [ REPLACE 'old_password' | EXPIRED ]
```

```

| [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE | EXPIRED }
| [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] |
DISABLE }
| VALID BEGIN 'timestamp'
| VALID UNTIL 'timestamp'

| USER GROUP 'groupuser'
| PERM SPACE 'spacelimit'
| TEMP SPACE 'tmpspacelimit'
| SPILL SPACE 'spillspacelimit'
| NODE GROUP logic_cluster_name
| PGUSER

```

- **Rename a role.**
ALTER ROLE role_name
RENAME TO new_name;
- **Lock or unlock.**
ALTER ROLE role_name
ACCOUNT { LOCK | UNLOCK };
- **Set parameters for a role.**
ALTER ROLE role_name [IN DATABASE database_name]
SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT};
- **Reset parameters for a role.**
ALTER ROLE role_name
[IN DATABASE database_name] RESET {configuration_parameter|ALL};

Parameters

- **role_name**
Specifies a role name.
Value range: an existing role name. If a role name contains uppercase letters, enclose the name with double quotation marks ("").
- **IN DATABASE database_name**
Modifies the parameters of a role in a specified database.
- **SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT}**
Sets parameters for a role. Session parameters modified by ALTER ROLE apply to a specified role and take effect in the next session triggered by the role.

NOTICE

The current version does not support setting user-level parameters.

Value range:

For details about the values of **configuration_parameter** and **value**, see [SET](#).

DEFAULT: clears the value of **configuration_parameter**.

configuration_parameter will inherit the default value of the new session generated for the role.

FROM CURRENT: uses the value of **configuration_parameter** of the current session.

- **RESET {configuration_parameter|ALL}**
Clears the value of **configuration_parameter**. The statement has the same effect as that of **SET configuration_parameter TO DEFAULT**.

NOTICE

The current version does not support the resetting of user-level parameters.

Value range: **ALL** indicates that the values of all parameters are cleared.

- **ACCOUNT LOCK | ACCOUNT UNLOCK**

- **ACCOUNT LOCK**: locks an account to forbid login to databases.
- **ACCOUNT UNLOCK**: unlocks an account to allow login to databases.

- **PGUSER**

In the current version, the PGUSER permission of a role cannot be modified.

- **PASSWORD/IDENTIFIED BY 'password'**

Resets or changes the user password. Except the initial user, other administrators and common users need to enter the correct old password when changing their own passwords. Only the initial user or users with the SYSADMIN or CREATEROLE permission can reset the password of a common user without entering the old password. The initial user can reset passwords of system administrators. System administrators cannot reset passwords of other system administrators.

- **EXPIRED**

Invalidate the password. Only the initial user or users with the SYSADMIN or CREATEROLE permission can invalidate user passwords. A system administrator can invalidate their own passwords or the passwords of other system administrators. The password of the initial user cannot be invalidated.

The user whose password is invalid can log in to the database but cannot perform the query operation. The query operation can be performed only after the password is changed or the administrator resets the password.

For details about other parameters, see [Parameters](#) in section "CREATE ROLE."

Examples

See [Examples](#) in section "CREATE ROLE."

Helpful Links

[CREATE ROLE](#), [DROP ROLE](#), and [SET ROLE](#)

7.14.22 ALTER ROW LEVEL SECURITY POLICY

Description

Modifies an existing row-level security policy, including the policy name and the users and expressions affected by the policy.

Precautions

Only the table owner or a system administrator can perform this operation.

Syntax

Change the name of an existing row-level security policy.

```
ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name RENAME TO new_policy_name;
```

Change the specified user and policy expression of an existing row-level security policy.

```
ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
  [ TO { role_name | PUBLIC } [, ...] ]
  [ USING ( using_expression ) ];
```

Parameters

- policy_name**
Specifies the name of a row-level security policy.
- table_name**
Specifies the name of a table to which a row-level security policy is applied.
- new_policy_name**
Specifies the new name of a row-level security policy.
- role_name**
Specifies names of users affected by a row-level security policy. PUBLIC indicates that the row-level security policy will affect all users.
- using_expression**
Specifies a row-level security policy, which is similar to a Boolean expression in the WHERE clause.

Examples

```
-- Create the data table all_data.
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

-- Create a row-level security policy to specify that the current user can view only their own data.
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);
gaussdb=# \d+ all_data
          Table "public.all_data"
Column |          Type          | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
id     | integer               |           |         |              |
role   | character varying(100) |           | extended |              |
data   | character varying(100) |           | extended |              |
Row Level Security Policies:
  POLICY "all_data_rls" FOR ALL
    TO public
    USING ((role)::name = "current_user"())
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no

-- Create users alice and bob.
gaussdb=# CREATE ROLE alice WITH PASSWORD "*****";
gaussdb=# CREATE ROLE bob WITH PASSWORD "*****";

-- Change the name of the all_data_rls policy.
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME TO all_data_new_rls;

-- Change the users affected by the row-level security policy.
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO alice, bob;
gaussdb=# \d+ all_data
          Table "public.all_data"
```

```

Column |      Type      | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
id     | integer        |           | plain   |              |
role   | character varying(100) |           | extended |              |
data   | character varying(100) |           | extended |              |
Row Level Security Policies:
  POLICY "all_data_new_rls" FOR ALL
  TO alice,bob
  USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Modify the expression defined for the row-level security policy.
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USING (id > 100 AND role =
current_user);
gaussdb=# \d+ all_data
              Table "public.all_data"
Column |      Type      | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
id     | integer        |           | plain   |              |
role   | character varying(100) |           | extended |              |
data   | character varying(100) |           | extended |              |
Row Level Security Policies:
  POLICY "all_data_new_rls" FOR ALL
  TO alice,bob
  USING (((id > 100) AND ((role)::name = "current_user"()))))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Delete users alice and bob.
gaussdb=# DROP ROLE alice, bob;

-- Delete the policy.
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data;

-- Delete the all_data table.
gaussdb=# DROP TABLE all_data;

```

Helpful Links

[CREATE ROW LEVEL SECURITY POLICY](#) and [DROP ROW LEVEL SECURITY POLICY](#)

7.14.23 ALTER SCHEMA

Description

Modifies schema attributes.

Precautions

- Only the owner of a schema or users granted with the ALTER permission on the schema can run the **ALTER SCHEMA** command. System administrators have this permission by default. To change the owner of a schema, you must be the owner of the schema or system administrator and a member of the new owner role.
- For system schemas other than public, such as pg_catalog and sys, only the initial user is allowed to change the owner of a schema. Changing the names of the built-in system schemas may make some functions unavailable or even

affect the normal running of the database. By default, the names of the built-in system schemas cannot be changed. To ensure forward compatibility, you can change the names of the built-in system schemas only when the system is being started or upgraded or when **allow_system_table_mods** is set to **on**.

- Only the initial user can change the owner of a schema to an O&M administrator. Other users cannot change the owner of a schema to an O&M administrator.

Syntax

- Rename a schema.

```
ALTER SCHEMA schema_name  
  RENAME TO new_name;
```
- Change the owner of a schema.

```
ALTER SCHEMA schema_name  
  OWNER TO new_owner;
```

Parameters

- **schema_name**
Specifies the name of an existing schema.
Value range: an existing schema name.
- **RENAME TO new_name**
Rename a schema.
new_name: new name of the schema.

NOTICE

- The schema name must be unique in the current database.
- The schema name cannot be the same as the initial username of the current database.
- The schema name cannot start with `pg_`.
- The schema name cannot start with `gs_role_`.

Value range: a string. It must comply with the [naming convention](#).

- **OWNER TO new_owner**
Change the owner of a schema. To do this as a non-administrator, you must be a direct or indirect member of the new owner role, and that role must have the CREATE permission on the database.
new_owner: new owner of the schema.
Value range: an existing username or role name.

Examples

```
-- Create the ds schema.  
gaussdb=# CREATE SCHEMA ds;  
  
-- Rename the current schema ds to ds_new.  
gaussdb=# ALTER SCHEMA ds RENAME TO ds_new;  
  
-- Create user jack.
```

```
gaussdb=# CREATE USER jack PASSWORD '*****';  
  
-- Change the owner of ds_new to jack.  
gaussdb=# ALTER SCHEMA ds_new OWNER TO jack;  
  
-- Delete user jack and schema ds_new.  
gaussdb=# DROP SCHEMA ds_new;  
gaussdb=# DROP USER jack;
```

Helpful Links

[CREATE SCHEMA](#) and [DROP SCHEMA](#)

7.14.24 ALTER SEQUENCE

Description

Modifies the parameters of an existing sequence.

Precautions

- Only the owner of a sequence, a user granted the ALTER permission on a sequence, or a user granted the ALTER ANY SEQUENCE permission on a sequence can run the **ALTER SEQUENCE** command. System administrators have this permission by default. To modify a sequence owner, you must be the sequence owner or system administrator and a member of the new owner role.
- In the current version, you can modify only the owner, owning column, and maximum value. To modify other parameters, delete the sequence and create it again. Then, use the Setval function to restore parameter values.
- ALTER SEQUENCE MAXVALUE cannot be used in transactions, functions, and stored procedures.
- After the maximum value of a sequence is changed, the cache of the sequence in all sessions is cleared.
- The ALTER SEQUENCE statement blocks the calling of nextval, setval, curval, and lastval.

Syntax

- Change the owning column and maximum value of a sequence.
ALTER SEQUENCE [IF EXISTS] name
[MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
[OWNED BY { table_name.column_name | NONE }] ;
- Change the owner of a sequence.
ALTER SEQUENCE [IF EXISTS] name OWNER TO new_owner;

Parameters

- **name**
Specifies the name of the sequence to be modified.
- **IF EXISTS**
Sends a notice instead of an error when you are modifying a nonexisting sequence.

- **MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE**
Specifies the maximum value of the sequence. The new maximum value must be greater than the current maximum value stored by GTM. If this parameter is not specified, the original maximum value is retained.
Value range: (gtm_last_value, $2^{63} - 1$].
- **OWNED BY**
Associates a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to.
If the sequence has been associated with another table before you use this option, the new association will overwrite the old one.
The associated table and sequence must be owned by the same user and in the same schema.
If **OWNED BY NONE** is used, all existing associations will be deleted.
- **new_owner**
Specifies the username of the new owner of the sequence. To change the owner, you must also be a direct or indirect member of the new role, and this role must have the CREATE permission on the sequence's schema.

Examples

```
-- Create an ascending sequence named serial, which starts from 101.
gaussdb=# CREATE SEQUENCE serial START 101;

-- Create a table and specify default values for the sequence.
gaussdb=# CREATE TABLE t1(c1 bigint default nextval('serial'));

-- Change the owning column of serial to T1.C1.
gaussdb=# ALTER SEQUENCE serial OWNED BY t1.c1;

-- Delete the sequence.
gaussdb=# DROP SEQUENCE serial cascade;
gaussdb=# DROP TABLE t1;
```

Helpful Links

[CREATE SEQUENCE](#) and [DROP SEQUENCE](#)

7.14.25 ALTER SERVER

Description

Adds, changes, or deletes the parameters of an existing server. You can query existing servers from the pg_foreign_server system catalog.

Precautions

Only the server owner or a user granted with the ALTER permission can run the **ALTER SERVER** command. System administrators have this permission by default. To change the owner of a server, the current user must be the owner of the server or system administrator, and the user must be a member of the new owner role.

When multi-layer quotation marks are used for sensitive columns (such as **password** and **secret_access_key**) in OPTIONS, the semantics is different from

that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

- Change the parameters for a foreign server.

```
ALTER SERVER server_name [ VERSION 'new_version' ]  
[ OPTIONS ( {[ ADD | SET | DROP ] option ['value']} [, ... ] ) ];
```

In **OPTIONS**, **ADD**, **SET**, and **DROP** are operations to be performed. If these operations are not specified, **ADD** operations will be performed by default. **option** and **value** are the parameters of the corresponding operation.

- Change the owner of a foreign server.

```
ALTER SERVER server_name  
OWNER TO new_owner;
```

- Change the name of a foreign server.

```
ALTER SERVER server_name  
RENAME TO new_name;
```

Parameters

The parameters for modifying the server are as follows:

- **server_name**

Specifies the name of the server to be modified.

- **new_version**

Specifies the source version of the server.

- **OPTIONS**

Changes options of the server. **ADD**, **SET**, and **DROP** are operations to be performed. If the operation is not set explicitly, **ADD** is used. The option name must be unique, and the name and value are also validated with the foreign data wrapper library of the server.

- **encrypt**

Specifies whether to encrypt data. This parameter can be set only when **type** is set to **OBS**. The default value is **off**.

Value range:

- **on** indicates that data is encrypted.
- **off** indicates that data is not encrypted.

- **access_key**

Specifies the access key (AK) (obtained by users from the OBS console) used for the OBS access protocol. When you create a foreign table. This parameter is available only when **type** is set to **OBS**.

- **secret_access_key**

Specifies the secret key (SK) value (obtained by users from the OBS console) used for the OBS access protocol. This parameter is available only when **type** is set to **OBS**.

- **new_owner**

Specifies the new owner of the server. To change the owner, you must be the owner of the foreign server and a direct or indirect member of the new owner

role, and must have the USAGE permission on the encapsulator of the foreign server.

- **new_name**
Specifies the new name of the server.

Examples

```
-- Create my_server.
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;

-- Change the name of an external service.
gaussdb=# ALTER SERVER my_server
  RENAME TO my_server_1;

-- Delete my_server_1.
gaussdb=# DROP SERVER my_server_1;
```

Helpful Links

[CREATE SERVER](#) and [DROP SERVER](#)

7.14.26 ALTER SESSION

Description

ALTER SESSION defines or modifies the conditions or parameters that affect the current session. Modified session parameters are kept until the current session is disconnected.

Precautions

- If the **START TRANSACTION** statement is not executed before the **SET TRANSACTION** statement, the transaction is ended instantly and the statement does not take effect.
- You can use the **transaction_mode(s)** method declared in the **START TRANSACTION** statement to avoid using the **SET TRANSACTION** statement. For details, see [START TRANSACTION](#).

Syntax

- Set transaction parameters of a session.

```
ALTER SESSION SET [ SESSION CHARACTERISTICS AS ] TRANSACTION
  { ISOLATION LEVEL { READ COMMITTED } | { READ ONLY | READ WRITE } } [, ...] ;
```
- Set other GUC parameters of a session.

```
ALTER SESSION SET
  {{config_parameter { { TO | = } { value | DEFAULT }
  | FROM CURRENT }} | CURRENT_SCHEMA [ TO | = ] { schema | DEFAULT }
  | TIME_ZONE time_zone
  | SCHEMA schema
  | NAMES encoding_name
  | ROLE role_name PASSWORD 'password'
  | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT }
  | XML OPTION { DOCUMENT | CONTENT }
  } ;
```

Parameters

- **config_parameter**

Specifies the name of the configurable GUC parameter. You can run the **SHOW ALL** command to view available GUC parameters.

 - **value**

Specifies the new value of **config_parameter**. You can specify the parameter as a string constant, identifier, number, or comma-separated list. **DEFAULT** is used to reset the parameter to its default value.

 - **DEFAULT**
 - **OFF**
 - **RESET**
 - User-specified value: The value must meet the restriction of the modified parameter.
 - **FROM CURRENT**

Uses the value of **configuration_parameter** of the current session.
- **CURRENT_SCHEMA schema**

Specifies the current schema.

Value range: an existing schema name. If the schema name does not exist, the value of **CURRENT_SCHEMA** will be empty.
- **TIME_ZONE timezone**

Specifies the local time zone for the current session.

Value range: a valid local time zone. The corresponding GUC parameter is **TimeZone**. The default value is **PRC**.
- **SCHEMA schema**

Specifies the current schema. Here the schema is a string.
- **NAMES encoding_name**

Specifies the client character encoding. This statement is equivalent to **set client_encoding to encoding_name**.

Value range: a valid character encoding name. The GUC parameter corresponding to this option is **client_encoding**. The default encoding is **UTF8**.
- **role_name**

Value range: a string. It must comply with the **naming convention**.
- **password**

Specifies the password of a role. It must comply with the password convention.
- **SESSION AUTHORIZATION**

Sets the session user identifier of the current session.
- **XML OPTION { DOCUMENT | CONTENT }**

Specifies the XML parsing mode.

Value range: **CONTENT** (default) and **DOCUMENT**.

Examples

```
-- Create the ds schema.
gaussdb=# CREATE SCHEMA ds;

-- Set the search path of a schema.
gaussdb=# SET SEARCH_PATH TO ds, public;

-- Set the time/date type to the traditional postgres format (date before month).
gaussdb=# SET DATESTYLE TO postgres, dmy;

-- Set the character code of the current session to UTF8.
gaussdb=# ALTER SESSION SET NAMES 'UTF8';

-- Set the time zone to Berkeley of California.
gaussdb=# SET TIME_ZONE 'PST8PDT';

-- Set the time zone to Italy.
gaussdb=# SET TIME_ZONE 'Europe/Rome';

-- Set the current schema.
gaussdb=# ALTER SESSION SET CURRENT_SCHEMA TO tpceds;

-- Set XML OPTION to DOCUMENT.
gaussdb=# ALTER SESSION SET XML_OPTION DOCUMENT;

-- Create the role joe, and set the session role to joe.
gaussdb=# CREATE ROLE joe WITH PASSWORD '*****';
gaussdb=# ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD '*****';

-- Switch to the default user.
gaussdb=> ALTER SESSION SET SESSION AUTHORIZATION default;

-- Delete the ds schema.
gaussdb=# DROP SCHEMA ds;

-- Delete the role joe.
gaussdb=# DROP ROLE joe;

-- Start a transaction and set the transaction level.
gaussdb=# START TRANSACTION;
gaussdb=# ALTER SESSION SET TRANSACTION READ ONLY;
gaussdb=# END;
```

Helpful Links

[SET](#)

7.14.27 ALTER SYNONYM

Description

Modifies the attributes of the SYNONYM object.

Precautions

- Currently, only the owner of the SYNONYM object can be changed.
- Only system administrators have the permission to modify the owner of the SYNONYM object.
- The new owner must have the CREATE permission on the schema where the SYNONYM object resides.

Syntax

```
ALTER SYNONYM synonym_name  
OWNER TO new_owner;
```

Parameters

- **synonym_name**
Specifies the name of the synonym to be modified, which can contain the schema name.
Value range: a string. It must comply with the [naming convention](#).
- **new_owner**
Specifies the new owner of the synonym object.
Value range: a string. It must be a valid username.

Examples

```
-- Create a system administrator.  
gaussdb=# CREATE USER sysadmin WITH SYSADMIN PASSWORD '*****';  
  
-- Switch to the system administrator.  
gaussdb=# \c - sysadmin  
  
-- Create synonym t1.  
gaussdb=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;  
  
-- Create user u1.  
gaussdb=# CREATE USER u1 PASSWORD '*****';  
  
-- Assign permissions to the new system administrator.  
gaussdb=# GRANT ALL ON SCHEMA sysadmin TO u1;  
  
-- Change the owner of synonym t1 to u1.  
gaussdb=# ALTER SYNONYM t1 OWNER TO u1;  
  
-- Delete synonym t1.  
gaussdb=# DROP SYNONYM t1;  
  
-- Revoke permissions from user u1.  
gaussdb=# REVOKE ALL ON SCHEMA sysadmin FROM u1;  
  
-- Delete user u1.  
gaussdb=# DROP USER u1;  
  
-- Switch to the initial user init_user. Replace init_user with the actual initial username.  
gaussdb=# \c - init_user  
  
-- Delete user sysadmin.  
gaussdb=# DROP USER sysadmin;
```

Helpful Links

[CREATE SYNONYM](#) and [DROP SYNONYM](#)

7.14.28 ALTER SYSTEM KILL SESSION

Description

ALTER SYSTEM KILL SESSION ends a session.

Precautions

None

Syntax

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [ IMMEDIATE ];
```

Parameters

- **session_sid, serial**

Specifies **SID** and **SERIAL** of a session (see examples for format). You can use the `pg_stat_activity` system catalog to query the current active threads (see the examples). However, when you run the **ALTER SYSTEM KILL SESSION** command, the threads may have ended.

Value range: SIDs and SERIALs of all sessions that can be queried from the system catalog `dv_sessions`

- **IMMEDIATE**

Specifies that a session will be ended instantly after the statement is executed.

Examples

```
-- Query session information.
gaussdb=# SELECT sid,serial#,username FROM dv_sessions;
   sid   | serial# | username
-----+-----+-----
140131075880720 | 0 | omm
140131025549072 | 0 | omm
140131073779472 | 0 | omm
140131071678224 | 0 | omm
140131125774096 | 0 |
140131127875344 | 0 |
140131113629456 | 0 |
140131094742800 | 0 |
(8 rows)

-- End the session whose SID is 140131075880720.
gaussdb=# ALTER SYSTEM KILL SESSION '140131075880720,0' IMMEDIATE;
```

7.14.29 ALTER TABLE

Description

Modifies tables, including modifying table definitions, renaming tables, renaming specified columns in tables, renaming table constraints, setting table schemas, enabling or disabling row-level security policies, and adding or updating multiple columns.

Precautions

- If the base table is a hash-distributed table, to create a primary key or unique index that does not contain the distribution key of the base table, use a GSI (by specifying the **BY GLOBAL INDEX** column); to create a primary key or unique index that contains the distribution key of the base table, use an ordinary index (without specifying the **BY GLOBAL INDEX** column). In single-DN deployment mode, both the GSI and ordinary index can be successfully

created. If the base table is a non-hash-distributed table, you can only create the primary key or unique index as an ordinary index. That is, the index key must contain the distribution key of the base table.

- The owner of a table, users granted with the ALTER permission on the table, or users granted with the ALTER ANY TABLE permission can run the **ALTER TABLE** command. System administrators have the permission to run the command by default. To modify the owner or schema of a table, you must be the table owner or system administrator and a member of the new owner role.
- The tablespace of a partitioned table cannot be modified, but the tablespace of the partition can be modified.
- The storage parameter **ORIENTATION** cannot be modified.
- Currently, SET SCHEMA can only set schemas to user schemas. It cannot set a schema to a system internal schema.
- The distribution key (or column) of a table cannot be modified.
- Auto-increment columns cannot be added, or a column whose **DEFAULT** value contains the nextval() expression cannot be added.
- Row-level security cannot be enabled for foreign tables and temporary tables.
- When you delete a PRIMARY KEY constraint by constraint name, the NOT NULL constraint is not deleted. If necessary, manually delete the NOT NULL constraint.
- When JDBC is used, the **DEFAULT** value can be set through PreparedStatement.
- If you add a column using ADD COLUMN, all existing rows in the table are initialized to the column's default value (**NULL** if no **DEFAULT** value is specified).

If no **DEFAULT** value is specified for the new column, **NULL** is used, and no full table update is triggered.

If the new column has the **DEFAULT** value, the column must meet all the following requirements. Otherwise, the entire table is updated, leading to additional overheads and affecting online services.

1. The data type is BOOL, BYTEA, SMALLINT, BIGINT, SMALLINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, or INTERVAL.

2. The length of the **DEFAULT** value of the added column cannot exceed 128 bytes.

3. The **DEFAULT** value of the added column does not contain the volatile function.

4. The **DEFAULT** value is required and cannot be **NULL**.

If you are not sure whether condition 3 is met, check whether the **provolatile** attribute of the function in the PG_RPOC system catalog is **v**.

- The number of table constraints cannot exceed 32,767.

Syntax

- Modify the definition of a table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
  action [, ... ];  
ALTER TABLE [ IF EXISTS ] table_name
```

```

ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint
[ ... ] ] } [, ...] );
ALTER TABLE [ IF EXISTS ] table_name
  MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
ALTER TABLE [ IF EXISTS ] table_name
  RENAME TO new_table_name;
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
  RENAME [ COLUMN ] column_name TO new_column_name;
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
  RENAME CONSTRAINT constraint_name TO new_constraint_name;
ALTER TABLE [ IF EXISTS ] table_name
  SET SCHEMA new_schema;

```

There are several clauses of **action**:

```

column_clause
| ADD table_constraint [ NOT VALID ]
| ADD table_constraint_using_index
| VALIDATE CONSTRAINT constraint_name
| DROP CONSTRAINT [ IF EXISTS ] constraint_name [ RESTRICT | CASCADE ]
| CLUSTER ON index_name
| SET WITHOUT CLUSTER
| SET ( {storage_parameter = value} [, ... ] )
| RESET ( storage_parameter [, ... ] )
| OWNER TO new_owner
| SET TABLESPACE new_tablespace
| SET {COMPRESS|NOCOMPRESS}
| TO { GROUP groupname | NODE ( nodename [, ... ] ) }
| ADD NODE ( nodename [, ... ] )
| DELETE NODE ( nodename [, ... ] )
| UPDATE SLICE LIKE table_name
| DISABLE TRIGGER [ trigger_name | ALL | USER ]
| ENABLE TRIGGER [ trigger_name | ALL | USER ]
| ENABLE REPLICA TRIGGER trigger_name
| ENABLE ALWAYS TRIGGER trigger_name
| ENABLE ROW LEVEL SECURITY
| DISABLE ROW LEVEL SECURITY
| FORCE ROW LEVEL SECURITY
| NO FORCE ROW LEVEL SECURITY

| REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
| AUTO_INCREMENT [ = ] value
| [ [ DEFAULT ] CHARACTER SET | CHARSET [ = ] default_charset ] [ [ DEFAULT ] COLLATE [ = ]
default_collation ]

```

 NOTE

- **ADD table_constraint [NOT VALID]**
Adds a table constraint.
- **ADD table_constraint_using_index**
Adds a primary key constraint or unique constraint to a table based on the existing unique index. If the specified index is a GSI, an error is reported. In this case, you need to use the BY GLOBAL INDEX syntax to add a GSI index constraint.
- **VALIDATE CONSTRAINT constraint_name**
Validates a check-class constraint created with the **NOT VALID** option, and scans the entire table to ensure that all rows meet the constraint. Nothing happens if the constraint is already marked valid.
- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**
Deletes a table constraint.
- **CLUSTER ON index_name**
Selects the default index for future CLUSTER operations. Actually, the table is not re-clustered.
- **SET WITHOUT CLUSTER**
Deletes the most recently used CLUSTER index from the table. This affects future CLUSTER operations that do not specify an index.
- **SET ({storage_parameter = value} [, ...])**
Changes one or more storage parameters for the table. If the value of **table_name** is an index name, **ACTIVE_PAGES** specifies the number of index pages, which may be less than the actual number of physical file pages and can be used for optimizer optimization. Currently, this parameter is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM and ANALYZE (including AUTOVACUUM). You are advised not to manually set this parameter because it is invalid in distributed mode.
- **RESET (storage_parameter [, ...])**
Resets one or more storage parameters to their defaults. As with SET, a table rewrite might be needed to update the table entirely.
- **OWNER TO new_owner**
Changes the owner of a table, sequence, or view to the specified user.
- **SET TABLESPACE new_tablespace**
Changes the table's tablespace to the specified tablespace and moves the data files associated with the table to the new tablespace. Indexes on the table, if any, are not moved; but they can be moved separately with additional **SET TABLESPACE** option in ALTER INDEX.
- **SET {COMPRESS|NOCOMPRESS}**
Sets the compression feature of a table. The table compression feature affects only the storage mode of data inserted in a batch subsequently and does not affect storage of existing data. Setting the table compression feature will result in the fact that there are both compressed and uncompressed data in the table. Row-store tables do not support compression.
- **TO { GROUP groupname | NODE (nodename [, ...]) }**
The syntax is only available in extended mode (when GUC parameter **support_extended_features** is on). Exercise caution when enabling the mode. It is mainly used for tools like internal dilatation tools. Common users should not use the mode. This command only modifies the logical mapping relationship of the table distribution nodes and does not migrate the table's metadata and data on the DN.
- **ADD NODE (nodename [, ...])**

It is only available for internal scale-out tools. Common users should not use the syntax.

- **DELETE NODE (nodename [, ...])**

It is only available for internal scale-in tools. Common users should not use the syntax.

- **UPDATE SLICE LIKE table_name**

This syntax is used by internal scaling tools and cannot be used by common users.

- **DISABLE TRIGGER [trigger_name | ALL | USER]**

Disables a single trigger specified by **trigger_name**, disables all triggers, or disables only user triggers (excluding internally generated constraint triggers, for example, deferrable unique constraint triggers and exclusion constraints triggers).

Exercise caution when using this function because data integrity cannot be ensured as expected if the triggers are not executed.

- **| ENABLE TRIGGER [trigger_name | ALL | USER]**

Enables a single trigger specified by **trigger_name**, enables all triggers, or enables only user triggers.

- **| ENABLE REPLICA TRIGGER trigger_name**

Determines that the trigger firing mechanism is affected by the configuration variable *session_replication_role*. When the replication role is **origin** (default value) or **local**, a simple trigger is fired.

When **ENABLE REPLICA** is configured for a trigger, it is triggered only when the session is in replica mode.

- **| ENABLE ALWAYS TRIGGER trigger_name**

Determines that all triggers are fired regardless of the current replica mode.

- **| { DISABLE | ENABLE } ROW LEVEL SECURITY**

Enables or disables row-level security for a table.

If row-level security is enabled for a data table but no row-level security policy is defined, the row-level access to the data table is not affected. If row-level security for a table is disabled, the row-level access to the table is not affected even if a row-level security policy has been defined. For details, see [CREATE ROW LEVEL SECURITY POLICY](#).

- **| { NO FORCE | FORCE } ROW LEVEL SECURITY**

Forcibly enables or disables row-level security for a table.

By default, the table owner is not affected by the row-level security feature. However, if row-level security is forcibly enabled, the table owner (excluding system administrators) will be affected. System administrators are not affected by any row-level security policies.

- **REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }**

Specifies the record level of old tuples in UPDATE and DELETE statements on a table in logical replication scenarios.

- **DEFAULT** records the old value of the primary key column. If there is no primary key, **DEFAULT** does not record the old value.
- **USING INDEX** records the old values of columns covered by the named indexes. These values must be unique, non-local, and non-deferrable, and contain the values of columns marked **NOT NULL**.
- **FULL** records the old values of all columns in the row.
- **NOTHING** does not record information in old rows.

In logical replication scenarios, when the UPDATE and DELETE statements of a table are parsed, the parsed old tuples consist of the information recorded in this method. For tables with primary keys, this option can be set to **DEFAULT** or **FULL**. For a table without a primary key, set this parameter to **FULL**. Otherwise, the old tuple will be parsed as empty during decoding. You are advised not to set this

parameter to **NOTHING** in common scenarios because old tuples are always parsed as empty.

- **AUTO_INCREMENT [=] value**
Sets the next auto-increment value of the auto-increment column. The configured value takes effect only when it is greater than the current auto-increment counter. The value must be a non-negative integer and cannot be greater than $2^{127} - 1$. This clause takes effect only when **sql_compatibility** is set to 'B'.
 - **[[DEFAULT] CHARACTER SET | CHARSET [=] default_charset] [[DEFAULT] COLLATE [=] default_collation]**
Modifies the default character set and default collation of a table to the specified values. The modification does not affect the existing columns in the table. This clause takes effect only when **sql_compatibility** is set to 'B'.
 - **CONVERT TO CHARACTER SET | CHARSET charset [COLLATE collation]**
Modifies the default character set and default collation of a table to the specified values, sets the character set and collation of all columns with character type to the specified value and converts the data in the column to new character set encoding.
- The **column_clause** can be one of the following clauses:
- ```

ADD [COLUMN] column_name data_type [CHARACTER SET | CHARSET charset]
[compress_mode] [COLLATE collation] [column_constraint [...]]
| MODIFY column_name data_type
| MODIFY column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE]
| MODIFY column_name [CONSTRAINT constraint_name] NULL
| DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]
| ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING
expression]
| ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }
| ALTER [COLUMN] column_name { SET | DROP } NOT NULL
| ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer
| ADD STATISTICS ((column_1_name, column_2_name [, ...]))
| DELETE STATISTICS ((column_1_name, column_2_name [, ...]))
| ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])
| ALTER [COLUMN] column_name RESET (attribute_option [, ...])
| ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }

```

 NOTE

- **ADD [ COLUMN ] column\_name data\_type [ compress\_mode ] [ COLLATE collation ] [ column\_constraint [ ... ] ]**  
Adds a column to a table. If a column is added using ADD COLUMN, all existing rows in the table are initialized with the column's default value (NULL if no DEFAULT clause is specified).
- **ADD ( { column\_name data\_type [ compress\_mode ] } [, ...] )**  
Adds columns in the table.
- **MODIFY ( { column\_name data\_type | column\_name [ CONSTRAINT constraint\_name ] NOT NULL [ ENABLE ] | column\_name [ CONSTRAINT constraint\_name ] NULL } [, ...] )**  
Modifies the data type of an existing column in the table. Running this command will clear the statistics of this column. You are advised to collect the statistics of this column again after the modification.
- **DROP [ COLUMN ] [ IF EXISTS ] column\_name [ RESTRICT | CASCADE ]**  
Drops a column from a table. Indexes and constraints related to the column are automatically dropped. If an object not belonging to the table depends on the column (for example, view), CASCADE must be specified.  
  
The DROP COLUMN statement does not physically remove the column, but simply makes it invisible to SQL operations. Subsequent INSERT and UPDATE operations in the table will store a **NULL** value for the column. Therefore, column deletion takes a short period of time but does not immediately release the tablespace on the disks, because the space occupied by the deleted column is not recycled. The space will be recycled when VACUUM is executed.
- **ALTER [ COLUMN ] column\_name [ SET DATA ] TYPE data\_type [ COLLATE collation ] [ USING expression ]**  
Modifies the type of a column in a table. Indexes and simple table constraints on the column will automatically use the new data type by reparsing the originally supplied expression.  
  
If the original data type of a column and the modified data type are binary compatible, you do not need to rewrite the entire table when running this statement. In other scenarios, the entire table is rewritten. You can check whether the original type and target type are binary compatible in the PG\_CAST system catalog. If **castmethod** is 'b', they are binary compatible. For example, if the data type of the source table is text and is converted to int, table rewriting is triggered. If it is converted to clob, table rewriting is not triggered. If table rewriting is triggered, the deleted space on the table is recycled immediately.  
  
Running this command will clear the statistics of this column. You are advised to collect the statistics of this column again after the modification.
- **ALTER [ COLUMN ] column\_name { SET DEFAULT expression | DROP DEFAULT }**  
Sets or removes the default value for a column. The default values only apply to subsequent INSERT operations; they do not cause rows already in the table to change. Defaults can also be created for views, in which case they are inserted into INSERT statements on the view before the view's ON INSERT rule is applied.
- **ALTER [ COLUMN ] column\_name { SET | DROP } NOT NULL**  
Changes whether a column is marked to allow null values or to reject null values. You can only use SET NOT NULL when the column contains no null values.
- **ALTER [ COLUMN ] column\_name SET STATISTICS [PERCENT] integer**

Specifies the per-column statistics-gathering target for subsequent ANALYZE operations. The target can be set in the range from 0 to 10000. Set it to **-1** to revert to using the default system statistics target.

- **{ADD | DELETE} STATISTICS ((column\_1\_name, column\_2\_name [, ...]))**  
Adds or deletes the declaration of collecting multi-column statistics to collect multi-column statistics as needed when ANALYZE is performed for a table or a database. The statistics about a maximum of 32 columns can be collected at a time. You are not allowed to add or delete such declaration for system catalogs or foreign tables.
- **{ENABLE | DISABLE } STATISTICS ((column\_1\_name, column\_2\_name [, ...]))**  
Enables or disables multi-column statistics. When automatic statistics creation is enabled (the GUC parameter **auto\_statistic\_ext\_columns** is required), you can disable specific multi-column combinations to prevent them from being automatically created and used.
- **ALTER [ COLUMN ] column\_name SET ( {attribute\_option = value} [, ... ] )**  
**ALTER [ COLUMN ] column\_name RESET ( attribute\_option [, ... ] )**  
Sets or resets per-attribute options.  
Currently, the only defined per-attribute options are **n\_distinct** and **n\_distinct\_inherited**. **n\_distinct** affects statistics of a table, while **n\_distinct\_inherited** affects the statistics of the table and its subtables. Currently, only **SET/RESET n\_distinct** is supported, and **SET/RESET n\_distinct\_inherited** is forbidden.
- **ALTER [ COLUMN ] column\_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**  
Sets the storage mode for a column. It specifies whether this column is held inline or in an attached table, and whether the data should be compressed. **SET STORAGE** does not change a table. It only specifies the recommended strategy for future table updates.

▪ **column\_constraint** is as follows:

```
[CONSTRAINT constraint_name]
{ NOT NULL |
 NULL |
 CHECK (expression) |
 DEFAULT default_expr |
 GENERATED ALWAYS AS (generation_expr) [STORED] |
 AUTO_INCREMENT |
 UNIQUE [KEY] index_parameters |
 PRIMARY KEY index_parameters |
 ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = column_encryption_key,
 ENCRYPTION_TYPE = encryption_type_value) |
 REFERENCES reftable [(refcolumn)] [MATCH FULL | MATCH PARTIAL | MATCH
SIMPLE]
 [ON DELETE action] [ON UPDATE action] }
[DEFERRABLE | NOT DEFERRABLE][INITIALLY DEFERRED | INITIALLY IMMEDIATE]
```

○ **index\_parameters** is as follows:

```
[WITH ({storage_parameter = value} [, ...])]
[USING INDEX TABLESPACE tablespace_name]
```

▪ **compress\_mode** of a column is as follows:

```
{ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
```

- **table\_constraint\_using\_index** used to add the primary key constraint or unique constraint based on the unique index is as follows:

```
[CONSTRAINT constraint_name]
{ UNIQUE | PRIMARY KEY } USING INDEX index_name
[DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE]
```

- **table\_constraint** is as follows:

```
[CONSTRAINT constraint_name]
{ CHECK (expression) }
```

```

UNIQUE (column_name [, ...]) index_parameters |
PRIMARY KEY (column_name [, ...]) index_parameters
}
[DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE]

```

- **index\_parameters** is as follows:

```

[WITH ({storage_parameter = value} [, ...])]
[USING INDEX TABLESPACE tablespace_name][BY GLOBAL INDEX]

```

#### NOTE

If **index\_parameters** is set to **BY GLOBAL INDEX**, the global secondary index is used to create constraints.

- Rename a table. The renaming does not affect stored data.

```

ALTER TABLE [IF EXISTS] table_name
 RENAME TO new_table_name;

```

- Rename the specified column in the table.

```

ALTER TABLE [IF EXISTS] { table_name [*] | ONLY table_name | ONLY (table_name) }
 RENAME [COLUMN] column_name TO new_column_name;

```

- Rename the constraint of the table.

```

ALTER TABLE [IF EXISTS] { table_name [*] | ONLY table_name | ONLY (table_name) }
 RENAME CONSTRAINT constraint_name TO new_constraint_name;

```

- Set the schema of the table.

```

ALTER TABLE [IF EXISTS] table_name
 SET SCHEMA new_schema;

```

#### NOTE

- The schema setting moves the table into another schema. Associated indexes and constraints owned by table columns are migrated as well. Currently, the schema for sequences cannot be changed. If the table has sequences, delete the sequences, and create them again or delete the ownership between the table and sequences. In this way, the table schema can be changed.
- To change the schema of a table, you must also have the CREATE permission on the new schema. To add the table as a new child of a parent table, you must own the parent table as well. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have the CREATE permission on the table's schema. These restrictions enforce that the user can only rebuild and delete the table. However, a system administrator can alter the ownership of any table anyway.
- All the actions except for RENAME and SET SCHEMA can be combined into a list of multiple alterations to apply in parallel. For example, it is possible to add several columns or alter the type of several columns in a single statement. This is useful with large tables, since only one pass over the tables need be made.
- Adding a CHECK or NOT NULL constraint will scan the table to validate that existing rows meet the constraint.
- Adding a column with a non-NULL default or changing the type of an existing column will rewrite the entire table. Rewriting a large table may take much time and temporarily needs doubled disk space.
- Add columns.

```

ALTER TABLE [IF EXISTS] table_name
 ADD ({ column_name data_type [compress_mode] [COLLATE collation] [column_constraint
[...] } } [, ...]);

```
- Update columns.

```

ALTER TABLE [IF EXISTS] table_name
 MODIFY ({ column_name data_type | column_name [CONSTRAINT constraint_name] NOT NULL
[ENABLE] | column_name [CONSTRAINT constraint_name] NULL } [, ...]);

```

## Parameters

- **IF EXISTS**  
Sends a notice instead of an error if no tables have identical names. The notice prompts that the table you are querying does not exist.
- **table\_name [\*] | ONLY table\_name | ONLY ( table\_name )**  
**table\_name** is the name of the table that you need to modify.  
If **ONLY** is specified, only the table is modified. If **ONLY** is not specified, the table and all subtables are modified. You can add the asterisk (\*) option following the table name to specify that all subtables are scanned, which is the default operation.
- **constraint\_name**  
Specifies the name of an existing constraint to drop.
- **index\_name**  
Specifies the index name.

---

### NOTICE

In the ADD CONSTRAINT operation:

- **index\_name** is supported only in B-compatible databases (that is, **sql\_compatibility** set to 'B').
- For foreign key constraints, if **constraint\_name** and **index\_name** are specified at the same time, **constraint\_name** is used as the index name.
- For a unique key constraint, if both **constraint\_name** and **index\_name** are specified, **index\_name** is used as the index name.

- 
- **USING method**  
Specifies the name of the index method to be used.  
For details about the value range, see [USING method](#).

---

### NOTICE

In the ADD CONSTRAINT operation:

- The USING method is supported only in B-compatible databases (that is, **sql\_compatibility** set to 'B').
- In B-compatible mode, if USING method is not specified, the default index method is btree for Astore or UB-tree for Ustore.

- 
- **ASC | DESC**  
**ASC** specifies an ascending (default) sort order. **DESC** specifies a descending sort order.

---

### NOTICE

In ADD CONSTRAINT, ASC|DESC is supported only in B-compatible databases (that is, **sql\_compatibility** set to 'B').

---

- **expression**  
Specifies an expression index constraint based on one or more columns of the table. It must be written in parentheses.

---

**NOTICE**

Expression indexes in the UNIQUE constraint are supported only in B-compatible databases (that is, **sql\_compatibility** set to 'B').

- **storage\_parameter**  
Specifies the name of a storage parameter.  
The following options are added for online scaling:
  - **append\_mode** (enumerated type)  
Scales out a table online or offline, or stops scaling it. You can modify certain content in the table during online scaling but cannot do so during offline scaling.  
To modify a table that is being scaled, append new data so that they can be recorded as incremental data.
    - **on**: scales out a table online. New data will be appended.
    - **off**: stops scaling. New data will be written in normal mode, and options for online scaling will not be displayed in **pg\_class.reloptions**.
    - **read\_only**: scales a table offline, during which no other operations can be performed on the table.
    - **end\_catchup**: reports errors for the write service in the last round of data increment. The read service is executed normally.
  - **rel\_cn\_oid** (OID type)  
Records the OID of tables on the current CN to generate **delete\_delta** on the DNs.  
If **append\_mode** is set to **on**, **rel\_cn\_oid** must be specified.  
The **append\_mode** and **rel\_cn\_oid** options are used only for online scale-out tools.
  - **exec\_step** (integer)  
Records resumable transmission steps in **relOptions** of the temporary table.  
Value range: [1,4]  
It can be used only for data redistribution.
  - **create\_time** (long integer)  
Records the time when the temporary table is created during resumable transmission in **relOptions** of the temporary table.  
Only the data redistribution tool is supported.
  - **wait\_clean\_cbi** (string type)  
Specifies whether the current global index contains the residual tuple generated during bucket migration for scaling. After scaling,

**wait\_clean\_cbi** is set to **y**. After the residual tuple is cleared in the vacuum process, **wait\_clean\_cbi** is set to **n**.

This option is used only in scaling tools.

- **redistribute\_version** (long integer)

Specifies the number of times that a table is successfully redistributed during logical scale-out. The value is used by the standby node to read and verify consistency. The value is incremented by 1 each time the table is redistributed.

This option is set only in the scale-out process and is not recommended.

The following option is added for creating an index:

- **parallel\_workers** (int type)

Number of bgworker threads started when an index is created. For example, value **2** indicates that two bgworker threads are started to create indexes concurrently.

Value range: [0,32]. The value 0 indicates that concurrent index creation is disabled.

Default value: If this parameter is not set, the concurrent index creation function is disabled.

The following option is added to the replication table:

- **primarynode** (Boolean type)

Default value: **off**

When **primarynode** is set to **on**, the primary node is selected for the replication table. Generally, the primary node is the first node recorded in the **nodeoids** column in the **pgxc\_class** table. When the IUD operation is performed on the replication table, the operation is delivered to the primary node first. After the result is received, the operation is delivered to other DNs.

- **logical\_repl\_node** (string type)

Name of the DN that returns logical logs to the CN during logical decoding of a distributed replication table. For the replication table, if this parameter is not specified, the first node in the node group where the current table is located is used by default. When the RESET operation is performed on this option, the node is reset to the first node of current table.

Value range: a string

Default value: For the non-replication table, this parameter is empty by default. For the replication table, this parameter is set to the name of the first node by default.

- **new\_owner**

Specifies the name of the new table owner.

- **new\_tablespace**

Specifies the new name of the tablespace to which the table belongs.

- **column\_name, column\_1\_name, column\_2\_name**

Specifies the name of a new or existing column.

- **data\_type**

Specifies the type of a new column or a new type of an existing column.

- **compress\_mode**  
Specifies whether to compress a table column. The clause specifies the compression algorithm preferentially used by the column. Row-store tables do not support compression.
- **collation**  
Specifies the collation rule name of a column. The optional COLLATE clause specifies a collation for the new column; if omitted, the collation is the default for the new column. You can run the **select \* from pg\_collation** command to query collation rules from the **pg\_collation** system catalog. The default collation rule is the row starting with **default** in the query result.
- **USING expression**  
Specifies how to compute the new column value from the old; if omitted, the default conversion is an assignment cast from old data type to new. A USING clause must be provided if there is no implicit or assignment cast from the old to new type.

 **NOTE**

USING in ALTER TYPE can specify any expression involving the old values of the row; that is, it can refer to any columns other than the one being cast. This allows general casting to be done with the ALTER TYPE syntax. Because of this flexibility, the USING expression is not applied to the column's default value (if any); the result might not be a constant expression as required for a default. This means that when there is no implicit or assignment cast from old to new type, ALTER TYPE might fail to convert the default even though a USING clause is supplied. In such cases, drop the default with DROP DEFAULT, perform ALTER TYPE, and then use SET DEFAULT to add a suitable new default. Similar considerations apply to indexes and constraints involving the column.

- **NOT NULL | NULL**  
Sets whether the column allows null values.
- **ENABLE**  
Specifies that the constraint is enabled. By default, the constraint is enabled.
- **integer**  
Specifies the constant value of a signed integer. When using **PERCENT**, the range of **integer** is from 0 to 100.
- **attribute\_option**  
Specifies an attribute option.
- **PLAIN | EXTERNAL | EXTENDED | MAIN**  
Specifies a column-store mode.
  - **PLAIN** must be used for fixed-length values (such as integers). It must be inline and uncompressed.
  - **MAIN** is for inline, compressible data.
  - **EXTERNAL** is for external, uncompressed data. Use of **EXTERNAL** will make substring operations on **text** and **bytea** values run faster, at the penalty of increased storage space.
  - **EXTENDED** is for external, compressed data. **EXTENDED** is the default for most data types that support non-**PLAIN** storage.
- **CHECK ( expression )**

New rows or rows to be updated must satisfy for an expression to be true. If any row produces a false result, an error is raised and the database is not modified.

A check constraint specified as a column constraint should reference only the column's values, while an expression in a table constraint can reference multiple columns.

Currently, **CHECK ( expression )** does not include subqueries and cannot use variables apart from the current column.

- **DEFAULT default\_expr**

Assigns a default data value to a column.

The data type of the default expression must match the data type of the column.

The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is **NULL**.

- **COLUMN\_ENCRYPTION\_KEY = column\_encryption\_key**

Specifies the name of the column encryption key in the ENCRYPTED WITH constraint.

Value range: a string. It must comply with the naming convention.

- **ENCRYPTION\_TYPE = encryption\_type\_value**

For the encryption type in the ENCRYPTED WITH constraint, the value of **encryption\_type\_value** is **DETERMINISTIC** or **RANDOMIZED**.

- **UNIQUE [KEY] index\_parameters**

**UNIQUE ( column\_name [, ... ] ) index\_parameters**

**UNIQUE** specifies that a group of one or more columns of a table can contain only unique values.

**UNIQUE KEY** can be used only when **sql\_compatibility** is set to **'MYSQL'**, which has the same semantics as **UNIQUE**.

- **PRIMARY KEY index\_parameters**

**PRIMARY KEY ( column\_name [, ... ] ) index\_parameters**

Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

- **DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE**

Sets whether the constraint can be deferrable.

- **DEFERRABLE**: deferrable to the end of the transaction and checked using **SET CONSTRAINTS**.
- **NOT DEFERRABLE**: checks immediately after the execution of each command.
- **INITIALLY IMMEDIATE**: checks immediately after the execution of each statement.
- **INITIALLY DEFERRED**: checks when the transaction ends.

- **WITH ( {storage\_parameter = value} [, ... ] )**

Specifies an optional storage parameter for a table or an index.

- **tablespace\_name**  
Specifies the name of the tablespace where the index locates.
- **COMPRESS|NOCOMPRESS**
  - **NOCOMPRESS**: If the **NOCOMPRESS** keyword is specified, the existing compression feature of the table will not be changed.
  - **COMPRESS**: If the **COMPRESS** keyword is specified, the table compression feature will be triggered by batch tuple insertion. Row-store tables do not support compression.
- **new\_table\_name**  
Specifies the new table name.
- **new\_column\_name**  
Specifies the new name of a specific column in a table.
- **new\_constraint\_name**  
Specifies the new name of a table constraint.
- **new\_schema**  
Specifies the new schema name.
- **CASCADE**  
Automatically drops objects that depend on the dropped column or constraint (for example, views referencing the column).
- **RESTRICT**  
Refuses to drop the column if the column is referenced by other columns or constraints. **RESTRICT** is the default option. If **CASCADE** is not specified, the value is **RESTRICT**. An example of the statement is as follows:  

```
alter table <Table name>[drop [column] <Column name> [cascade | restrict]];
```
- **schema\_name**  
Specifies the schema name of a table.

## Examples of Modifying a Table

- Rename a table.  

```
gaussdb=# CREATE TABLE aa(c1 int, c2 int);
gaussdb=# ALTER TABLE IF EXISTS aa RENAME TO test_alt1;
```
- Modify the schema of a table.  

```
-- Create the test_schema schema.
gaussdb=# CREATE SCHEMA test_schema;

-- Change the schema of the test_alt1 table to test_schema.
gaussdb=# ALTER TABLE test_alt1 SET SCHEMA test_schema;

-- Query table information.
gaussdb=# SELECT schemaname,tablename FROM pg_tables WHERE tablename = 'test_alt1';
schemaname | tablename
-----+-----
test_schema | test_alt1
(1 row)
```
- Change the owner of a table.  

```
-- Create user test_user.
gaussdb=# CREATE USER test_user PASSWORD 'XXXXXXXXXX';

-- Change the owner of the test_alt1 table to test_user.
gaussdb=# ALTER TABLE IF EXISTS test_schema.test_alt1 OWNER TO test_user;
```

```
-- Query.
gaussdb=# SELECT tablename, schemaname, tableowner FROM pg_tables WHERE tablename =
'test_alt1';
tablename | schemaname | tableowner
-----+-----+-----
test_alt1 | test_schema | test_user
(1 row)
```

- **Modify the tablespace of a table.**

```
-- Create the tbs_data1 tablespace.
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';

-- Change the tablespace of the test_alt1 table to tbs_data1.
gaussdb=# ALTER TABLE test_schema.test_alt1 SET TABLESPACE tbs_data1;

-- Query.
gaussdb=# SELECT tablename, tablespace FROM pg_tables WHERE tablename = 'test_alt1';
tablename | tablespace
-----+-----
test_alt1 | tbs_data1
(1 row)

-- Delete.
gaussdb=# DROP TABLE test_schema.test_alt1;
gaussdb=# DROP TABLESPACE tbs_data1;
gaussdb=# DROP SCHEMA test_schema;
gaussdb=# DROP USER test_user;
```

## Examples of Modifying a Column

- **Change column names.**

```
-- Create a table.
gaussdb=# CREATE TABLE test_alt2(c1 INT,c2 INT);

-- Change column names.
gaussdb=# ALTER TABLE test_alt2 RENAME c1 TO id;
gaussdb=# ALTER TABLE test_alt2 RENAME COLUMN c2 to areaid;

-- Query.
\d test_alt1
Table "public.test_alt1"
Column | Type | Modifiers
-----+-----+-----
id | integer |
areaid | integer |
```

- **Add columns.**

```
-- Add a column to the test_alt1 table.
gaussdb=# ALTER TABLE IF EXISTS test_alt2 ADD COLUMN name VARCHAR(20);

-- Query.
gaussdb=# \d test_alt2
Table "public.test_alt1"
Column | Type | Modifiers
-----+-----+-----
id | integer |
areacode | integer |
name | character varying(20) |
```

- **Modify the data type of a column.**

```
-- Change the type of the name column in the test_alt1 table.
gaussdb=# ALTER TABLE test_alt1 MODIFY name VARCHAR(50);

-- Query.
gaussdb=# \d test_alt1
Table "public.test_alt2"
Column | Type | Modifiers
-----+-----+-----
id | integer |
```

```

areaid | integer |
name | character varying(50) |
-- Change the type of the name column in the test_alt1 table.
gaussdb=# ALTER TABLE test_alt2 ALTER COLUMN name TYPE VARCHAR(25);

-- Query.
gaussdb=# \d test_alt2
 Table "public.test_alt2"
Column | Type | Modifiers
-----+-----+-----
id | integer |
areaid | integer |
name | character varying(25) |

```

- **Delete a column.**

```

-- Delete the areaid column from test_alt1.
gaussdb=# ALTER TABLE test_alt2 DROP COLUMN areaid;

-- Query.
gaussdb=# \d test_alt2
 Table "public.test_alt2"
Column | Type | Modifiers
-----+-----+-----
id | integer |
name | character varying(25) |

```

- **Modify the column-store mode.**

```

-- View table details.
gaussdb=# \d+ test_alt2
 Table "public.test_alt2"
Column | Type | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
id | integer | | plain | |
name | character varying(25) | | extended | |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE

-- Change the storage mode of the name column in the test_alt2 table.
gaussdb=# ALTER TABLE test_alt2 ALTER COLUMN name SET STORAGE PLAIN;

-- Query.
gaussdb=# \d+ test_alt2
 Table "public.test_alt2"
Column | Type | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
id | integer | | plain | |
name | character varying(25) | | plain | |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE

-- Delete.
gaussdb=# DROP TABLE test_alt2;

```

## Examples of Modifying a Constraint

- **Add a not-null constraint to a column.**

```

-- Create a table.
gaussdb=# CREATE TABLE test_alt3(pid INT, areaid CHAR(5), name VARCHAR(20));

Add a not-null constraint to pid.
gaussdb=# ALTER TABLE test_alt3 MODIFY pid NOT NULL;

-- Query.
gaussdb=# \d test_alt3
 Table "public.test_alt3"
Column | Type | Modifiers
-----+-----+-----
pid | integer | not null
areaid | character(5) |
name | character varying(20) |

```

- Cancel the not-null constraint on a column.

```
gaussdb=# ALTER TABLE test_alt3 MODIFY pid NULL;
```

```
-- Query.
```

```
gaussdb=# \d test_alt3
 Table "public.test_alt3"
Column | Type | Modifiers
-----+-----+-----
 pid | integer |
 areaid | character(5) |
 name | character varying(20) |
```

- Modify the default value of a column.

```
-- Modify the default value of id in the test_alt1 table.
```

```
gaussdb=# ALTER TABLE test_alt3 ALTER COLUMN areaid SET DEFAULT '00000';
```

```
-- Query.
```

```
gaussdb=# \d test_alt3
 Table "public.test_alt3"
Column | Type | Modifiers
-----+-----+-----
 pid | integer |
 areaid | character(5) | default '00000'::bpchar
 name | character varying(20) |
```

```
-- Delete the default value of id.
```

```
gaussdb=# ALTER TABLE test_alt3 ALTER COLUMN areaid DROP DEFAULT;
```

```
-- Query.
```

```
gaussdb=# \d test_alt3
 Table "public.test_alt3"
Column | Type | Modifiers
-----+-----+-----
 pid | integer |
 areaid | character(5) |
 name | character varying(20) |
```

- Add a table-level constraint.

- Directly add a constraint.

```
-- Add a primary key constraint to the table.
```

```
gaussdb=# ALTER TABLE test_alt3 ADD CONSTRAINT pk_test3_pid PRIMARY KEY (pid);
```

```
-- Query.
```

```
gaussdb=# \d test_alt3
 Table "public.test_alt3"
Column | Type | Modifiers
-----+-----+-----
 pid | integer | not null
 areaid | integer |
 name | character varying(20) |
Indexes:
 "pk_test3_pid" PRIMARY KEY, btree (pid) TABLESPACE pg_default
```

- Create an index and then add constraints.

```
-- Create a table.
```

```
gaussdb=# CREATE TABLE test_alt4(c1 INT, c2 INT);
```

```
-- Create an index.
```

```
gaussdb=# CREATE UNIQUE INDEX pk_test4_c1 ON test_alt4(c1);
```

```
-- Associate the created index when adding a constraint.
```

```
gaussdb=# ALTER TABLE test_alt4 ADD CONSTRAINT pk_test4_c1 PRIMARY KEY USING INDEX pk_test4_c1;
```

```
-- Query.
```

```
gaussdb=# \d test_alt4
 Table "public.test_alt4"
Column | Type | Modifiers
-----+-----+-----
 c1 | integer | not null
 c2 | integer |
Indexes:
```

- ```
"pk_test4_c1" PRIMARY KEY, btree (c1) TABLESPACE pg_default
-- Delete.
gaussdb=# DROP TABLE test_alt4;
```
- Delete a table-level constraint.
-- Delete a constraint.
gaussdb=# ALTER TABLE test_alt3 DROP CONSTRAINT IF EXISTS pk_test3_pid;
-- Query.
gaussdb=# \d test_alt3
Table "public.test_alt3"
Column | Type | Modifiers
-----+-----+-----
pid | integer | not null
areaid | integer |
name | character varying(20) |
-- Delete.
gaussdb=# DROP TABLE test_alt3;

Helpful Links

[CREATE TABLE](#) and [DROP TABLE](#)

7.14.30 ALTER TABLE PARTITION

Description

Modifies table partitions, including adding, deleting, splitting, merging, clearing, swapping, and renaming partitions, moving partition tablespaces, and modifying partition attributes.

Precautions

- Only the partitioned table owner or a user granted with the ALTER permission can run the **ALTER TABLE PARTITION** command. System administrators have this permission by default.
- The tablespace of the added partition cannot be PG_GLOBAL.
- The name of the added partition must be different from names of existing partitions in the partitioned table.
- The key value of the added partition must be consistent with the type of partition keys in the partitioned table.
- If a range partition is added, the key value of the added partition must be greater than the upper limit of the last range partition in the partitioned table.
- If a list partition is added, the key value of the added partition cannot be the same as that of an existing partition.
- Hash partitions cannot be added.
- If the number of partitions in the target partitioned table has reached the maximum (**1048575**), partitions cannot be added.
- If a partitioned table has only one partition, the partition cannot be deleted.
- Use PARTITION FOR() to choose partitions. The number of specified values in the brackets should be the same as the column number in customized partition, and they must be consistent.

- The **Value** partitioned table does not support the ALTER PARTITION operation.
- Hash partitioned tables do not support splitting, combination, addition, and deletion of partitions.
- Deleting, splitting, merging, clearing, and exchanging partitions will invalidate global indexes. The UPDATE GLOBAL INDEX clause can be used to update the indexes synchronously.
- If the UPDATE GLOBAL INDEX clause is not used when you delete, split, merge, clear, or exchange partitions, concurrent DML services may report errors due to invalidated indexes.
- Deleting, splitting, merging, clearing, and exchanging partitions will invalidate global secondary indexes in partitioned tables. For exchanging partitions, all global secondary indexes in ordinary tables will also become invalid. Concurrent DML services may report errors due to unavailable indexes.
- If **enable_gpi_auto_update** is set to **on**, the global index is automatically updated even if the UPDATE GLOBAL INDEX clause is not declared.

Syntax

Modifying a partition in a partitioned table includes modifying the table partition itself and the table partition name, and resetting the partition ID.

- Modify the syntax of the table partition.
`ALTER TABLE [IF EXISTS] { table_name [*] | ONLY table_name | ONLY (table_name) }
action [, ...];`

action indicates the following clauses for maintaining partitions. For the partition continuity when multiple clauses are used for partition maintenance, GaussDB executes DROP PARTITION and ADD PARTITION first, and then the rest clauses in sequence.

```
move_clause |  
exchange_clause |  
row_clause |  
merge_clause |  
modify_clause |  
split_clause |  
add_clause |  
drop_clause
```

- The `move_clause` syntax is used to move the partition to a new **tablespace**.
`MOVE PARTITION { partition_name | FOR (partition_value [, ...]) } TABLESPACE tablespacename`
- The `exchange_clause` syntax is used to move the data from an ordinary table to a specified partition.
`EXCHANGE PARTITION { (partition_name) | FOR (partition_value [, ...]) }
WITH TABLE {[ONLY] ordinary_table_name | ordinary_table_name * | ONLY
(ordinary_table_name)}
[{ WITH | WITHOUT } VALIDATION] [VERBOSE] [UPDATE GLOBAL INDEX]`

The ordinary table and partition whose data is to be exchanged must meet the following requirements:

- The number of columns of the ordinary table is the same as that of the partition, and their information should be consistent, including: column name, data type, constraint, collation information, storage parameter, and compression information.

- The compression information of the ordinary table and partition should be consistent.
- The distribution key information of the ordinary table and partition should be consistent.
- The number of ordinary table indexes is the same as that of local indexes of the partition, and the index information is the same.
- The number and information of constraints of the ordinary table and partition should be consistent.
- The ordinary table cannot be a temporary table.
- When the built-in security policy is enabled, an ordinary table cannot contain columns bound to a dynamic data anonymization policy.

NOTICE

- When the exchange is done, the data and tablespace of the ordinary table and partition are exchanged. The statistics of the ordinary table and partition are no longer inaccurate after the exchange, and they should be analyzed again.
- A non-partition key cannot be used to create a local unique index. Therefore, if an ordinary table contains a unique index, data cannot be exchanged.
To exchange data, you can create an intermediate table, insert partition data into the intermediate table, truncate partitions, insert ordinary table data into the partitioned table, drop the ordinary table, and rename the intermediate table.
- If the DROP COLUMN operation is performed on an ordinary or partitioned table, the deleted column still exists physically. Therefore, you need to ensure that the deleted column of the ordinary table is strictly aligned with that of the partition.

-
- The row_clause syntax is used to set row movement of a partitioned table.
`{ ENABLE | DISABLE } ROW MOVEMENT`
 - The merge_clause syntax is used to merge partitions into one. The maximum number of source partitions that can be merged in a command is 300.
`MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
[TABLESPACE tablespacename] [UPDATE GLOBAL INDEX]`

NOTICE

For range partitioning and interval partitioning, the ranges of the source partitions must increase continuously, and the partition name after MERGE can be the same as the name of the last source partition. For list partitioning, there is no such range requirement on the source partitions, and the partition name after MERGE can be the same as that of any source partition. If the partition name after MERGE is the same as that of a source partition, they are considered as the same partition.

NOTICE

Ustore tables do not support ALTER TABLE MERGE PARTITIONS in transaction blocks and stored procedures.

- The `modify_clause` syntax is used to set whether a partitioned index is usable.

```
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
```

- The `split_clause` syntax is used to split one partition into partitions.

```
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_point_clause | no_split_point_clause } [ UPDATE GLOBAL INDEX ]
```

NOTICE

- The partition name after SPLIT can be the same as the source partition name, but they are regarded as different partitions.
-

- The `split_point_clause` syntax for specifying the split point for a range partitioned table is as follows:

```
AT ( partition_value ) INTO ( PARTITION partition_name [ TABLESPACE tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
```

NOTICE

The size of the split point should be in the range of partition keys of the partition to be split. The split point can only split one partition into two new partitions.

- The `no_split_point_clause` syntax without specifying a split point for a range partitioned table is as follows:

```
INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
```

NOTICE

- The first new partition key specified by **partition_less_than_item** should be greater than that of the previously split partition (if any), and the last partition key specified by **partition_less_than_item** should equal that of the partition being split.
- The first new partition key specified by **partition_start_end_item** should equal that of the former partition (if any), and the last partition key specified by **partition_start_end_item** should equal that of the partition being split.
- **partition_less_than_item** supports a partition key with up to 16 columns, while **partition_start_end_item** supports a one-column partition key. For details about the supported data types, see [•PARTITION BY RANGE \[COLUMNS\] \(partition_key\)](#).
- **partition_less_than_item** and **partition_start_end_item** cannot be used in the same statement. There is no restriction on different SPLIT statements.

- The **partition_less_than_item** syntax is as follows:

```
PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE } [, ...] )  
[ TABLESPACE tablespacename ]
```
- The **partition_start_end_item** syntax is as follows. For details about the constraints, see [START END](#).

```
PARTITION partition_name {  
  {START(partition_value) END (partition_value) EVERY (interval_value)} |  
  {START(partition_value) END ({partition_value | MAXVALUE})} |  
  {START(partition_value)} |  
  {END({partition_value | MAXVALUE})}  
} [TABLESPACE tablespacename]
```
- The syntax for specifying the split point for a list partitioned table is as follows:

```
VALUES ( partition_value_list ) INTO ( PARTITION partition_name [ TABLESPACE  
tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
```

NOTICE

The split point must be a non-empty true subset of the source partition. Specifying a split point can only split one partition into two partitions.

- The syntax for not specifying the split point for a list partitioned table is as follows:

```
INTO ( PARTITION partition_name VALUES (partition_value_list) [ TABLESPACE  
tablespacename ][, ...] )
```

NOTICE

- The range of the last partition is not defined; the partition range is equal to the remaining set of the source partition excluding other level-2 partitions.
- If no split point is specified, each new partition must be a non-empty true subset of the source partition and does not overlap with each other.

-
- The `add_clause` syntax is used to add one or more partitions to a specified partitioned table.

```
ADD {partition_less_than_item | partition_start_end_item}
```

The `partition_list_item` syntax is as follows:

```
PARTITION partition_name VALUES (list_values_clause)  
[ TABLESPACE tablespacename ]
```

NOTICE

- **partition_list_item** supports a maximum of 16 partition keys. For details about the supported data types, see [•PARTITION BY LIST \[COLUMNS\] \(partition_key\)](#).
- Partitions cannot be added to a hash partitioned table.

-
- The `drop_clause` syntax is used to remove a partition from a specified partitioned table.

```
DROP PARTITION { partition_name | FOR ( partition_value [, ...] ) } [ UPDATE GLOBAL INDEX ]
```

NOTICE

- Hash partitioned table does not support partition deletion.
- If a partitioned table has only one partition, the partition cannot be deleted.

-
- The `truncate_clause` syntax is used to remove a specified partition from a partitioned table.

```
TRUNCATE PARTITION { partition_name | FOR ( partition_value [, ...] ) } [ UPDATE GLOBAL INDEX ]
```

- The syntax for modifying the name of a partition is as follows:

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
  RENAME PARTITION { partition_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```
- Reset a partition ID.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) } RESET  
PARTITION;
```

Parameters

- **table_name**
Specifies the name of a partitioned table.
Value range: an existing table name
- **partition_name**

Specifies the name of a partition.

Value range: an existing partition name

- **tablespacename**

Specifies which tablespace the partition moves to.

Value range: an existing tablespace name

- **partition_value**

Specifies the key value of a partition.

The value specified by **PARTITION FOR (partition_value [, ...])** can uniquely identify a partition.

Value range: partition keys for the partition to be operated.

- **UNUSABLE LOCAL INDEXES**

Sets all the indexes unusable in the partition.

- **REBUILD UNUSABLE LOCAL INDEXES**

Rebuilds all the indexes in the partition.

- **{ ENABLE | DISABLE } ROW MOVEMET**

Sets row movement.

If the tuple value is updated on the partition key during the **UPDATE** action, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

Value range:

- **ENABLE**: Row movement is enabled.
- **DISABLE**: Row movement is disabled.

By default, this parameter is disabled.

- **ordinary_table_name**

Specifies the name of the ordinary table whose data is to be migrated.

Value range: an existing table name

- **{ WITH | WITHOUT } VALIDATION**

Checks whether the ordinary table data meets the specified partition key range of the partition to be migrated.

Value range:

- **WITH**: checks whether the ordinary table data meets the partition key range of the partition to be migrated. If any data does not meet the required range, an error is reported.
- **WITHOUT**: does not check whether the ordinary table data meets the partition key range of the partition to be migrated.

The default value is **WITH**.

The check is time consuming, especially when the data volume is large.

Therefore, use **WITHOUT** when you are sure that the current ordinary table data meets the partition key range of the partition to be migrated.

- **VERBOSE**

When **VALIDATION** is **WITH**, if the ordinary table contains data that is out of the partition key range, insert the data to the correct partition. If there is no correct partition where the data can be inserted to, an error is reported.

NOTICE

Only when **VALIDATION** is **WITH, VERBOSE** can be specified.

- **partition_new_name**
Specifies the new name of a partition.
Value range: a string. It must comply with the [naming convention](#).
- **UPDATE GLOBAL INDEX**
If this parameter is used, all global indexes in a partitioned table are updated to ensure that correct data can be queried using global indexes.
If this parameter is not used, all global indexes in a partitioned table will become invalid.

Examples

See [Examples](#) in section "CREATE TABLE PARTITION."

Helpful Links

[CREATE TABLE PARTITION](#) and [DROP TABLE](#)

7.14.31 ALTER TABLESPACE

Description

Modifies the attributes of a tablespace.

Precautions

- The ALTER TABLESPACE syntax cannot be used in the current version.
- Only the tablespace owner or a user granted with the ALTER permission can run the **ALTER TABLESPACE** command. System administrators have this permission by default. To modify a tablespace owner, you must be the tablespace owner or system administrator and a member of the new owner role.
- The ALTER TABLESPACE operation on a row-store table cannot be performed in a transaction block.
- To change the owner, you must also be a direct or indirect member of the new owning role.

NOTE

If **new_owner** is the same as **old_owner**, the current user will not be verified. A message indicating successful ALTER execution is displayed.

Syntax

- The syntax of renaming a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name  
  RENAME TO new_tablespace_name;
```
- The syntax of setting the owner of a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name  
OWNER TO new_owner;
```

- The syntax of setting the attributes of a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name  
SET ( { tablespace_option = value } [, ... ] );
```
- The syntax of resetting the attributes of a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name  
RESET ( { tablespace_option } [, ... ] );
```
- The syntax of setting the quota of a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name  
RESIZE MAXSIZE { UNLIMITED | 'space_size'};
```

Parameters

- **tablespace_name**
Specifies the tablespace to be modified.
Value range: an existing tablespace name.
- **new_tablespace_name**
Specifies the new name of a tablespace.
The new name cannot start with PG_.
Value range: a string. It must comply with the [naming convention](#).
- **new_owner**
Specifies the new owner of the tablespace.
Value range: an existing username.
- **tablespace_option**
Sets or resets the parameters of a tablespace.
Value range:
 - **seq_page_cost**: sets the optimizer to calculate the cost of obtaining disk pages in sequence. The default value is **1.0**.
 - **random_page_cost**: sets the optimizer to calculate the cost of obtaining disk pages in a non-sequential manner. The default value is **4.0**.

NOTE

- The value of **random_page_cost** is relative to that of **seq_page_cost**. It is meaningless when the value is equal to or less than the value of **seq_page_cost**.
- The prerequisite for the default value **4.0** is that the optimizer uses indexes to scan table data and the hit ratio of table data in the cache is about 90%.
- If the size of the table data space is smaller than that of the physical memory, decrease the value to a proper level. On the contrary, if the hit ratio of table data in the cache is lower than 90%, increase the value.
- If random-access memory like SSD is adopted, the value can be decreased to a certain degree to reflect the cost of true random scan.

Value range: Positive number of the floating-point type.

- **RESIZE MAXSIZE**
Resets the maximum size of tablespace.
Value range:
 - **UNLIMITED**: No limit is set for the tablespace.

- Determined by **space_size**. For details about the format, see [CREATE TABLESPACE](#).

 NOTE

- If the adjusted quota is smaller than the current tablespace usage, the adjustment is successful. You need to decrease the tablespace usage to a value less than the new quota before writing data to the tablespace.
- You can also use the following statement to change the value of **MAXSIZE**:

```
ALTER TABLESPACE tablespace_name RESIZE MAXSIZE  
{ 'UNLIMITED' | 'space_size'};
```

Examples

See [Examples](#) in section "CREATE TABLESPACE."

Helpful Links

[CREATE TABLESPACE](#) and [DROP TABLESPACE](#)

7.14.32 ALTER TRIGGER

Description

Renames a trigger.

 NOTE

Currently, only the name can be modified.

Precautions

The owner of the table where a trigger resides or a user granted the ALTER ANY TRIGGER permission can perform the ALTER TRIGGER operation. A system administrator has this permission by default.

Syntax

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

Parameters

- **trigger_name**
Specifies the name of the trigger to be modified.
Value range: an existing trigger
- **table_name**
Specifies the name of the table where the trigger to be modified is located.
Value range: an existing table having a trigger
- **new_name**
Specifies the new name after modification.
Value range: a string, which complies with the [naming convention](#). A value contains a maximum of 63 characters and cannot be the same as other triggers on the same table.

Examples

For details, see [CREATE TRIGGER](#).

Helpful Links

[CREATE TRIGGER](#), [DROP TRIGGER](#), and [ALTER TABLE](#)

7.14.33 ALTER TYPE

Description

Modifies the definition of a type.

Precautions

Only the owner of a type, a user granted the ALTER permission on a type, or a user granted the ALTER ANY TYPE permission on a sequence can run the **ALTER TYPE** command. System administrators have this permission by default. To modify the owner or schema of a type, you must be a type owner or system administrator and a member of the new owner role.

Syntax

- **Modify a type.**
`ALTER TYPE name action [, ...];`
The clauses corresponding to **action** are as follows:
 - **Add an attribute to a composite type.**
`ADD ATTRIBUTE attribute_name data_type [COLLATE collation] [CASCADE | RESTRICT]`
 - **Delete an attribute from a composite type.**
`DROP ATTRIBUTE [IF EXISTS] attribute_name [CASCADE | RESTRICT]`
 - **Change the type of an attribute in a composite type.**
`ALTER ATTRIBUTE attribute_name [SET DATA] TYPE data_type [COLLATE collation] [CASCADE | RESTRICT]`
- **Change the owner of a type.**
`ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER };`
- **Change the name of a type.**
`ALTER TYPE name RENAME TO new_name;`
- **Move a type to a new schema.**
`ALTER TYPE name SET SCHEMA new_schema;`
- **Add a new value to an enumerated type.**
`ALTER TYPE name ADD VALUE [IF NOT EXISTS] new_enum_value [{ BEFORE | AFTER } neighbor_enum_value];`
- **Change an enumerated value in the value list.**
`ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value;`

Parameters

- **name**
Specifies the name of an existing type that needs to be modified (optionally schema-qualified).
- **new_name**

- Specifies the new name of the type.
- **new_owner**
Specifies the new owner of the type.
 - **new_schema**
Specifies the new schema of the type.
 - **attribute_name**
Specifies the name of the attribute to be added, modified, or deleted.
 - **new_attribute_name**
Specifies the new name of the attribute to be renamed.
 - **data_type**
Specifies the data type of the attribute to be added, or the new type of the attribute to be modified.
 - **new_enum_value**
Specifies a new enumerated value. It is a non-null string with a maximum length of 64 bytes.
 - **neighbor_enum_value**
Specifies an existing enumerated value before or after which a new enumerated value will be added.
 - **existing_enum_value**
Specifies an enumerated value to be changed. It is a non-null string with a maximum length of 64 bytes.
 - **CASCADE**
Determines that the type to be modified, its associated records, and subtables that inherit the type will all be updated.
 - **RESTRICT**
Refuses to update the associated records of the modified type. This is the default action.

NOTICE

- **ADD ATTRIBUTE, DROP ATTRIBUTE, and ALTER ATTRIBUTE** can be combined for processing. For example, it is possible to add several attributes or change the types of several attributes at the same time in one command.
- To modify a schema of a type, you must have the **CREATE** permission on the new schema. To alter the owner, you must be a direct or indirect member of the new owner role, and that member must have **CREATE** permission on the schema of this type (these restrictions enforce that the alter owner will not do anything that cannot be done by deleting and rebuilding the type). However, system administrators can modify the rights of any type in any way. To add an attribute or modify the type of an attribute, you must also have the **USAGE** permission of this type.

-
- **CURRENT_USER**
Specifies the current user.

- **SESSION_USER**
Specifies the current system user.
- **COLLATE collation**
Assigns a collation to the column, which must be a sortable data type. If the collation is not specified, the default collation for the column's data type is used.

Examples

See [Examples](#) in section "CREATE TYPE."

Helpful Links

[CREATE TYPE](#) and [DROP TYPE](#)

7.14.34 ALTER USER

Description

Alters the attributes of a database user.

Precautions

Session parameters modified by ALTER USER apply to a specified user and take effect in the next session.

Syntax

- Modify user permissions or other information.

```
ALTER USER user_name [ [ WITH ] option [ ... ] ];
```

The option clause is as follows:

```
{ CREATEDB | NOCREATEDB }
| { CREATEROLE | NOCREATEROLE }
| { INHERIT | NOINHERIT }
| { AUDITADMIN | NOAUDITADMIN }
| { SYSADMIN | NOSYSADMIN }
| { MONADMIN | NOMONADMIN }
| { OPRADMIN | NOOPRADMIN }
| { POLADMIN | NOPOLADMIN }
| { USEFT | NOUSEFT }
| { LOGIN | NOLOGIN }
| { REPLICATION | NOREPLICATION }
| { VCADMIN | NOVCADMIN }
| { PERSISTENCE | NOPERSISTENCE }
| CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' [ EXPIRED ] | DISABLE | EXPIRED }
| [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' |
EXPIRED ] | DISABLE }
| VALID BEGIN 'timestamp'
| VALID UNTIL 'timestamp'

| USER GROUP 'groupuser'
| PERM SPACE 'spacelimit'
| TEMP SPACE 'tmpspacelimit'
| SPILL SPACE 'spillspacelimit'
| NODE GROUP logic_cluster_name
| PGUSER
```

- Change the username.

```
ALTER USER user_name  
  RENAME TO new_name;
```

- Lock or unlock.

```
ALTER USER user_name  
  ACCOUNT { LOCK | UNLOCK };
```

Parameters

- **user_name**

Specifies the current username.

Value range: an existing username. If a username contains uppercase letters, enclose the name with double quotation marks ("").

- **new_password**

Specifies a new password.

The new password must:

- Differ from the old password.
- Contain at least eight characters. This is the default length.
- Differ from the username or the username spelled backward.
- Contain at least three types of the following four types of characters: uppercase characters (A to Z), lowercase characters (a to z), digits (0 to 9), and special characters, including: ~!@#%&*()-_+=\|[]{};:;<.>/? If the password contains characters other than the preceding characters, an error will be reported during statement execution.

Value range: a string.

- **old_password**

Specifies the old password.

- **ACCOUNT { LOCK | UNLOCK }**

- **ACCOUNT LOCK:** locks an account to forbid login to databases.
- **ACCOUNT UNLOCK:** unlocks an account to allow login to databases.

- **PGUSER**

In the current version, the **PGUSER** attribute of a user cannot be modified.

For details about other parameters, see "Parameters" in [CREATE ROLE](#) and [ALTER ROLE](#).

NOTICE

The current version does not support the setting of user-level parameters.

Examples

```
-- Create user jim whose login password is *****  
gaussdb=# CREATE USER jim PASSWORD '*****';
```

```
-- Change the login password of user jim. (Use the ALTER USER... IDENTIFIED BY... REPLACE... syntax to  
change the user password. The password before REPLACE is the new password, and the password after  
REPLACE is the original password.)
```

```
gaussdb=# ALTER USER jim IDENTIFIED BY '*****' REPLACE '*****';
```

```
-- Set enable_seqscan to on. (The setting will take effect in the next session.)
gaussdb=# ALTER USER jim SET enable_seqscan TO on;

-- Reset the enable_seqscan parameter for jim.
gaussdb=# ALTER USER jim RESET enable_seqscan;

-- Lock jim.
gaussdb=# ALTER USER jim ACCOUNT LOCK;

-- Unlock jim.
gaussdb=# ALTER USER jim ACCOUNT UNLOCK;

-- Change the username.
gaussdb=# ALTER USER jim RENAME TO lisa;

-- Delete the user.
gaussdb=# DROP USER lisa CASCADE;
```

Helpful Links

[CREATE ROLE](#), [CREATE USER](#), and [DROP USER](#)

7.14.35 ALTER VIEW

Function

ALTER VIEW modifies all auxiliary attributes of a view. (To modify the query definition of a view, use **CREATE OR REPLACE VIEW**.)

Precautions

Only the view owner or a user granted with the ALTER permission can run the **ALTER VIEW** command. The system administrator has this permission by default. The following is permission constraints depending on attributes to be modified:

- To modify the schema of a view, you must be the owner of the view or system administrator and have the CREATE permission on the new schema.
- To modify the owner of a view, you must be the owner of the view or system administrator and a member of the new owner role, with the CREATE permission on the schema of the view.
- Do not change the type of a column in a view.

Syntax

- Set the default value of a view column.
ALTER VIEW [IF EXISTS] view_name
ALTER [COLUMN] column_name SET DEFAULT expression;
- Remove the default value of a view column.
ALTER VIEW [IF EXISTS] view_name
ALTER [COLUMN] column_name DROP DEFAULT;
- Change the owner of a view.
ALTER VIEW [IF EXISTS] view_name
OWNER TO new_owner;
- Rename a view.
ALTER VIEW [IF EXISTS] view_name
RENAME TO new_name;
- Set the schema of a view.

```
ALTER VIEW [ IF EXISTS ] view_name  
SET SCHEMA new_schema;
```

- **Set the options of a view.**

```
ALTER VIEW [ IF EXISTS ] view_name  
SET ( { view_option_name [ = view_option_value ] } [, ... ] );
```
- **Reset the options of a view.**

```
ALTER VIEW [ IF EXISTS ] view_name  
RESET ( view_option_name [, ... ] );
```

Parameter Description

- **IF EXISTS**
If this option is used, no error is generated when the view does not exist, and only a message is displayed.
- **view_name**
Specifies the view name, which can be schema-qualified.
Value range: a string. It must comply with the [naming convention](#).
- **column_name**
Specifies an optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.
Value range: a string. It must comply with the [naming convention](#).
- **SET/DROP DEFAULT**
Sets or deletes the default value of a column. This parameter does not take effect.
- **new_owner**
Specifies the new owner of a view.
- **new_name**
Specifies the new view name.
- **new_schema**
Specifies the new schema of the view.
- **view_option_name [= view_option_value]**
Specifies an optional parameter for a view.
 - **security_barrier**: specifies whether the view provides row-level security. The value is of the Boolean type. The default value is **true**.
- **expression**
Specifies constants, functions, or SQL expressions.

Examples

```
-- Create a schema.  
gaussdb=# CREATE SCHEMA tpcds;  
  
-- Create the tpcds.customer table.  
gaussdb=# CREATE TABLE tpcds.customer  
(  
c_customer_sk      INTEGER      NOT NULL,  
c_customer_id     CHARACTER(16) NOT NULL  
);  
  
-- Insert multiple records into the table.  
gaussdb=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAABAAAAAAA'),(100,
```

```
'AAAAAAAAAAAAAAAA'),(150, 'AAAAAAAAADAAAAAAAA');  
  
-- Create a view consisting of rows with c_customer_sk less than 150.  
gaussdb=# CREATE VIEW tpcds.customer_details_view_v1 AS  
SELECT * FROM tpcds.customer  
WHERE c_customer_sk < 150;  
  
-- Rename a view.  
gaussdb=# ALTER VIEW tpcds.customer_details_view_v1 RENAME TO customer_details_view_v2;  
  
-- Change the schema of a view.  
gaussdb=# ALTER VIEW tpcds.customer_details_view_v2 SET schema public;  
  
-- Delete a view.  
gaussdb=# DROP VIEW public.customer_details_view_v2;  
  
-- Delete the tpcds.customer table.  
gaussdb=# DROP TABLE tpcds.customer;  
  
-- Delete a schema.  
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[CREATE VIEW](#) and [DROP VIEW](#)

7.14.36 ANALYZE | ANALYSE

Description

- ANALYZE collects statistics about ordinary tables in a database and stores the results in the PG_STATISTIC and PG_STATISTIC_EXT system catalogs. After you run the **ANALYZE** command, you can query the collected statistics in the preceding system catalogs or system views PG_STATS and PG_EXT_STATS. The execution plan generator uses these statistics to generate the most effective execution plan.
- If no parameters are specified, ANALYZE analyzes each table and partitioned table in the database. You can also specify **table_name**, **column_name**, and **partition_name** to limit the analysis to a specified table, column, or partitioned table.
- ANALYZE | ANALYSE VERIFY checks whether data files of ordinary tables in a database are damaged.

Precautions

- Non-temporary tables cannot be analyzed in an anonymous block, transaction block, function, or stored procedure. Temporary tables in a stored procedure can be analyzed but their statistics updates cannot be rolled back.
- Remote read is not involved in the ANALYZE VERIFY scenario. Therefore, the remote read parameter does not take effect. If the system detects that a page is damaged due to an error in a key system catalog, the system directly reports an error and halts the detection.
- With no parameter specified, ANALYZE processes all the tables on which the current user has permission to analyze in the current database. With a table specified, ANALYZE processes only that table.
- To perform ANALYZE operation to a table, you must be a table owner or a user granted the VACUUM permission on the table. System administrators

have this permission by default. However, database owners are allowed to analyze all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true database-wide analyze operation can only be executed by system administrators). ANALYZE skips tables on which users do not have permissions.

- ANALYZE does not collect columns for which comparison or equivalent operations cannot be performed, for example, columns of the cursor type.
- If the table to be analyzed is empty, ANALYZE does not record the statistics of the table. The existing statistics are retained.

Syntax

- Collect statistics information about a table.

```
{ ANALYZE | ANALYSE } [ VERBOSE ]  
[ table_name [ ( column_name [, ...] ) ] ];
```

- Collect partition statistics on a partitioned table. This syntax is not supported currently.

```
{ ANALYZE | ANALYSE } [ VERBOSE ]  
table_name [ ( column_name [, ...] ) ] PARTITION ( partition_name );
```

NOTE

An ordinary partitioned table supports the syntax but not the function of collecting statistics about specified partitions.

- Collect statistics about global secondary indexes.

```
{ ANALYZE | ANALYSE } GLOBAL INDEX index_name FOR TABLE table_name;
```

NOTE

- Before performing ANALYZE on the global secondary index, you need to perform ANALYZE on the base table to ensure the accuracy of global secondary indexes.
 - After performing the REINDEX/REBUILD operation on the global secondary indexes, you need to perform ANALYZE on the base table first before the global secondary indexes to ensure the accuracy of global secondary indexes.
 - To collect statistics from global secondary indexes, you must specify **index_name** and **table_name**, and **index_name** must be the name of a global secondary index.
 - The global secondary indexes include only the statistics (reltuple and relpages) in the pg_class system catalog.
- Collect statistics about multiple columns manually.

```
{ ANALYZE | ANALYSE } [ VERBOSE ]  
table_name (( column_1_name, column_2_name [, ...] ));
```

NOTE

- The statistics about a maximum of 32 columns can be collected at a time.
 - You are not allowed to collect statistics about multiple columns in system catalogs.
- Collect statistics about multiple columns automatically.

After the **auto_statistic_ext_columns** parameter is enabled and the ANALYZE statement is executed, multi-column statistics are automatically created based on the index prefix of the table. The number of columns in the multi-column statistics cannot exceed the value of **auto_statistic_ext_columns**.

For example, if index (a, b, c, d) exists in table **t** and **auto_statistic_ext_columns** is set to **4**, multi-column statistics about (a, b), (a, b, c), and (a, b, c, d) are created after table **t** is analyzed.

```
{ ANALYZE | ANALYSE } [ VERBOSE ] table_name;
```

- Check the data files in the current database.

```
{ ANALYZE | ANALYSE } VERIFY { FAST | COMPLETE };
```

 **NOTE**

- In fast mode, DML operations need to be performed on the tables to be verified concurrently. As a result, an error is reported during the verification. In the current fast mode, data is directly read from the disk. When other threads modify files concurrently, the obtained data is incorrect. Therefore, you are advised to perform the verification offline.
- You can perform operations on the entire database. Because a large number of tables are involved, you are advised to save the results in redirection mode.

```
gsql -d database -p port -f sqlfile> sqllog.txt 2>&1
```
- Temporary tables and unlogged tables are not supported.
- NOTICE is used to check only tables that are visible to external systems. The detection of internal tables is included in the external tables on which NOTICE depends and is not displayed externally.
- This statement can be executed with error tolerance.
- If a key system catalog is damaged during a full database operation, an error is reported and the operation stops.

- Check data files of tables and indexes.

```
{ ANALYZE | ANALYSE } VERIFY { FAST | COMPLETE } { table_name | index_name } [ CASCADE ];
```

 **NOTE**

- Operations on ordinary tables and index tables are supported, but CASCADE operations on indexes of index tables are not supported. The CASCADE mode is used to process all index tables of the main table. When the index tables are checked separately, the CASCADE mode is not required.
- Temporary tables and unlogged tables are not supported.
- When the main table is checked, the internal tables of the main table, such as the TOAST table, are also checked.
- When the system displays a message indicating that the index table is damaged, you are advised to run the **reindex** command to rebuild the index.

- Check the data files of the table partition.

```
{ ANALYZE | ANALYSE } VERIFY { FAST | COMPLETE } table_name PARTITION (partition_name) [ CASCADE ];
```

 **NOTE**

- You can check a single partition of a table, but cannot perform the CASCADE operation on the indexes of an index table.
- Temporary tables and unlogged tables are not supported.

Parameters

- **VERBOSE**

Enables the display of progress messages.

 **NOTE**

If **VERBOSE** is specified, ANALYZE displays the progress information, indicating the table that is being processed. Statistics about tables are also displayed.

- **table_name**

Specifies the name (possibly schema-qualified) of a specific table to analyze. If omitted, all regular tables (but not foreign tables) in the current database are analyzed.

Currently, you can use ANALYZE to collect statistics on foreign tables of row-store tables.

Value range: an existing table name.

- **column_name, column_1_name, column_2_name**

Specifies the name of a specific column to analyze. All columns are analyzed by default.

Value range: an existing column name

- **partition_name**

Assumes the table is a partitioned table. You can specify **partition_name** following the keyword **PARTITION** to analyze the statistics of this table.

Currently, ANALYZE can be performed on partitioned tables, but statistics of specified partitions cannot be analyzed.

Value range: a partition name of a table

- **index_name**

Specifies the name of the specific index table to be analyzed (possibly schema-qualified).

Value range: an existing table name

- **FAST|COMPLETE**

The **FAST** mode verifies the CRC and page header of the table. If the verification fails, an alarm is generated. In **COMPLETE** mode, the pointer and tuple of the table are parsed and verified.

- **CASCADE**

In **CASCADE** mode, all indexes of the current table are verified.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE customer_info
(
  WR_RETURNED_DATE_SK    INTEGER           ,
  WR_RETURNED_TIME_SK    INTEGER           ,
  WR_ITEM_SK             INTEGER           NOT NULL,
  WR_REFUNDED_CUSTOMER_SK INTEGER
)
DISTRIBUTE BY HASH (WR_ITEM_SK);
-- Create a partitioned table.
gaussdb=# CREATE TABLE customer_par
(
  WR_RETURNED_DATE_SK    INTEGER           ,
  WR_RETURNED_TIME_SK    INTEGER           ,
  WR_ITEM_SK             INTEGER           NOT NULL,
  WR_REFUNDED_CUSTOMER_SK INTEGER
)
DISTRIBUTE BY HASH (WR_ITEM_SK)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
  PARTITION P1 VALUES LESS THAN(2452275),
  PARTITION P2 VALUES LESS THAN(2452640),
  PARTITION P3 VALUES LESS THAN(2453000),
  PARTITION P4 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
-- Run the ANALYZE statement to update statistics.
gaussdb=# ANALYZE customer_info;
-- Run the ANALYZE VERBOSE statement to update statistics and display table information.
gaussdb=# ANALYZE VERBOSE customer_info;
```

```
INFO: analyzing "public.customer_info"(cn_5002 pid=53078)
INFO: analyzing "public.customer_info" inheritance tree(cn_5002 pid=53078)
ANALYZE
```

NOTE

If any environment-related fault occurs, check the logs of CN.

```
-- Delete the tables.
gaussdb=# DROP TABLE customer_info;
gaussdb=# DROP TABLE customer_par;
```

7.14.37 BEGIN

Description

BEGIN may be used to initiate an anonymous block or a single transaction. This section describes the syntax of BEGIN used to initiate an anonymous block. For details about the BEGIN syntax that initiates transactions, see [START TRANSACTION](#).

An anonymous block is a structure that can dynamically create and execute stored procedure code instead of permanently storing code as a database object in the database.

Precautions

None

Syntax

- Enable an anonymous block.

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```
- Start a transaction.

```
BEGIN [ WORK | TRANSACTION ]
[
{
ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
| { READ WRITE | READ ONLY }
} [, ...]
];
```

Parameters

- **declare_statements**
Declares a variable, including its name and type, for example, **sales_cnt int**.
- **execution_statements**
Specifies the statement to be executed in an anonymous block.
Value range: an existing function name

Examples

```
-- Generate a string using an anonymous block.
gaussdb=# BEGIN
dbe_output.print_line('Hello');
```

```
END;  
/
```

Helpful Links

[START TRANSACTION](#)

7.14.38 CALL

Description

Calls defined functions and stored procedures.

Precautions

The owner of a function or stored procedure, users granted with the EXECUTE permission on the function or stored procedure, or users granted with the EXECUTE ANY FUNCTION permission can call the function or stored procedure. System administrators have the permission to call the function or stored procedure by default.

Syntax

```
CALL [ schema.] { func_name | procedure_name } ( param_expr );
```

Parameters

- **schema**
Specifies the name of the schema where a function or stored procedure is located.
- **func_name**
Specifies the name of the function or stored procedure to be called.
Value range: an existing function name
 **NOTE**
You can use database links to perform operations on remote functions or stored procedures. For details, see [DATABASE LINK](#).
- **param_expr**
Specifies a list of parameters in the function. Use := or => to separate a parameter name and its value. This method allows parameters to be placed in any order. If only parameter values are in the list, the value order must be the same as that defined in the function or stored procedure.
Value range: an existing function parameter name or stored procedure parameter name
 **NOTE**
 - The parameters include input parameters (whose name and type are separated by **IN**) and output parameters (whose name and type are separated by **OUT**). When you run the **CALL** command to call a function or stored procedure, the parameter list must contain an output parameter for non-overloaded functions. You can set the output parameter to a variable or any constant. For details, see [Examples](#). If an output parameter is contained, it must be a constant.

Examples

```
-- Create a function func_add_sql, calculate the sum of two integers, and return the result.
gaussdb=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

-- Pass by parameter value.
gaussdb=# CALL func_add_sql(1, 3);

-- Pass by naming tag method.
gaussdb=# CALL func_add_sql(num1 => 1,num2 => 3);
gaussdb=# CALL func_add_sql(num2 := 2, num1 := 3);

-- Delete the function.
gaussdb=# DROP FUNCTION func_add_sql;

-- Create a function with output parameters.
gaussdb=# CREATE FUNCTION func_increment_sql(num1 IN integer, num2 IN integer, res OUT integer)
RETURN integer
AS
BEGIN
res := num1 + num2;
END;
/

-- Specify a constant as an output parameter.
gaussdb=# CALL func_increment_sql(1,2,1);

-- Specify a variable as an output parameter.
gaussdb=# DECLARE
res int;
BEGIN
func_increment_sql(1, 2, res);
dbe_output.print_line(res);
END;
/

-- Drop the function.
gaussdb=# DROP FUNCTION func_increment_sql;
```

Helpful Links

[CREATE FUNCTION](#) and [CREATE PROCEDURE](#)

7.14.39 CHECKPOINT

Function

A checkpoint is a point in the transaction log sequence at which all data files have been updated to reflect the information in the log. All data files will be flushed to a disk.

CHECKPOINT forces a transaction log checkpoint. By default, WALs periodically specify checkpoints in a transaction log. You may use **gs_guc** to specify run-time parameters **checkpoint_segments**, **checkpoint_timeout**, and **incremental_checkpoint_timeout** to adjust the atomized checkpoint intervals.

Precautions

- Only the system administrator and O&M administrator can invoke **CHECKPOINT**.
- **CHECKPOINT** forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system.

Syntax

```
CHECKPOINT;
```

Parameter Description

None

Examples

```
-- Set a checkpoint.  
gaussdb=# CHECKPOINT;
```

7.14.40 CLEAN CONNECTION

Description

Clear idle or invalid network connections between the current CN and other specified CNs or DNs. This statement is used to clear the specified database and idle or invalid connections of a specified user cached in the current CN on a specified CN.

Precautions

- In non-force mode, this function only clears connections between database cluster nodes (CNs/DNs) and does not affect client connections.
- This function clears only idle and invalid connections cached on the CN. Normal connections that are being used are not cleared.
- This function takes effect only on CNs and does not take effect on DNs.
- You can query the `PG_STAT_GET_POOLER_STATUS()` function to check the cache connection and verify the clearing effect.
- You are advised to perform this operation only when the network connection of the database is abnormal.

Syntax

```
CLEAN CONNECTION  
TO { COORDINATOR ( nodename [, ... ] ) | NODE ( nodename [, ... ] ) } ALL [ CHECK ] [ FORCE ] }  
{ FOR DATABASE dbname | TO USER username | FOR DATABASE dbname TO USER username };
```

Parameters

- **CHECK**
This parameter can be specified only when the node list is specified as **TO ALL**. Setting this parameter will check whether a database is accessed by other sessions before its connections are cleared. If any sessions are detected before **DROP DATABASE** is executed, an error will be reported and the database will not be deleted.

- **FORCE**

This parameter can be specified only when the node list is TO ALL. If this parameter is specified, all threads related to the specified dbname and username in the current CN receive the SIGTERM signal, the corresponding session is forcibly closed, the transaction is terminated, and the network connection is cleared.

- **COORDINATOR (nodename ,nodename ... }) | NODE (nodename , nodename ...) | ALL**

This command is used to delete the idle or invalid connections between the current CN node and a specified node. There are three scenarios:

- **COORDINATOR**: Delete the idle or invalid connections from the current CN to a specified CN.
- **NODE**: Delete the idle or invalid connections from the current CN to a specified DN.
- **ALL**: Delete the idle or invalid connections from the current CN to all nodes, including CNs and DNs.

Value range: **nodename** is an existing node name.

- **dbname**

Deletes connections to a specified database from the current CN. If this parameter is not specified, connections to all databases will be deleted.

Value range: an existing database name

- **username**

Deletes connections to a specified user from the current CN. If this parameter is not specified, connections of all users will be deleted.

Value range: an existing username

Examples

```
-- Create the test_clean_connection database.
gaussdb=# CREATE DATABASE test_clean_connection;

-- Create user jack.
gaussdb=# CREATE USER jack PASSWORD '*****';

-- Delete the idle and invalid connections between the current CN and dn1 and dn2 in the template1
database.
gaussdb=# CLEAN CONNECTION TO NODE (dn_6001_6002,dn_6003_6004) FOR DATABASE template1;

-- Delete the idle and invalid connections between the current CN and dn1 that are related to user jack.
gaussdb=# CLEAN CONNECTION TO NODE (dn_6001_6002) TO USER jack;

-- Drop the connections between the current CN and all nodes related to the test_clean_connection
database.
gaussdb=# CLEAN CONNECTION TO ALL FORCE FOR DATABASE test_clean_connection;

-- Delete user jack.
gaussdb=# DROP USER jack;

-- Drop the test_clean_connection database.
gaussdb=# DROP DATABASE test_clean_connection;
```

7.14.41 CLOSE

Function

CLOSE frees the resources associated with an open cursor.

Precautions

- After a cursor is closed, no subsequent operations are allowed on it.
- A cursor should be closed when it is no longer needed.
- Every non-holdable open cursor is implicitly closed when a transaction is terminated by **COMMIT** or **ROLLBACK**.
- A holdable cursor is implicitly closed if the transaction that created it aborts by **ROLLBACK**.
- If the cursor creation transaction is successfully committed, the holdable cursor remains open until an explicit **CLOSE** operation is executed, or the client disconnects.
- GaussDB does not have an explicit **OPEN** cursor statement. A cursor is considered open when it is declared. You can view all available cursors by querying the **pg_cursors** system view.

Syntax

```
CLOSE { cursor_name | ALL } ;
```

Parameter Description

- **cursor_name**
Specifies the name of a cursor to be closed.
- **ALL**
Closes all open cursors.

Examples

See [Examples](#) in **FETCH**.

Helpful Links

[FETCH](#) and [MOVE](#)

7.14.42 CLUSTER

Description

- Clusters a table based on an index.
- **CLUSTER** instructs GaussDB to cluster the table specified by **table_name** based on the index specified by **index_name**. The index must have been defined by **table_name**.
- When a table is clustered, it is physically reordered based on the index information. Clustering is a one-time operation. When the table is

subsequently updated, the changes are not clustered. That is, no attempt is made to store new or updated rows according to their index order.

- When a table is clustered, GaussDB records which index the table was clustered by. `CLUSTER table_name` reclusters the clustered index that was previously recorded in the table. You can also use `ALTER TABLE table_name CLUSTER` on `index_name` to set the index of a specified table for subsequent cluster operations, or use `ALTER TABLE table_name SET WITHOUT CLUSTER` to clear the previously clustered index of a specified table.
- If `CLUSTER` does not contain parameters, all tables that have been clustered in the database owned by the current user will be reprocessed. If a system administrator uses this command, all clustered tables are reclustered.
- When a table is being clustered, an `ACCESS EXCLUSIVE` lock is acquired on it. This prevents any other database operations (both read and write) from being performed on the table until the `CLUSTER` operation is finished.

Precautions

- Only row-store B-tree indexes support `CLUSTER`.
- In the case where you are accessing single rows randomly within a table, the actual order of the data in the table is unimportant. However, if there are many accesses to some data and an index groups the data, using the `CLUSTER` index improves performance.
- If an index you request from a table is a range, or a single indexed value that has multiple rows that match, `CLUSTER` will help because once the index identifies the table page for the first row that matches, all other rows that match are probably already on the same table page, and so you save disk accesses and speed up the query.
- During clustering, the system creates a temporary backup of the table created in the index sequence and a temporary backup of each index in the table. Therefore, ensure that the disk has sufficient free space during clustering, which is at least the sum of the table size and all index sizes.
- `CLUSTER` records which indexes have been used for clustering. Therefore, you can manually specify indexes for the first time, cluster specified tables, and set a maintenance script that will be executed periodically. You only need to run the **`CLUSTER`** command without parameters. In this way, tables that you want to periodically cluster can be automatically updated.
- The optimizer records table clustering statistics. After clustering a table, you need to execute the `ANALYZE` operation to ensure that the optimizer has the latest clustering information. Otherwise, the optimizer may select a non-optimal query plan.
- `CLUSTER` cannot be executed in transactions.
- If the `xc_maintenance_mode` parameter is not enabled, `CLUSTER` skips all system catalogs.

Syntax

- Cluster a table.
`CLUSTER [VERBOSE] table_name [USING index_name];`
- Cluster a partition.
`CLUSTER [VERBOSE] table_name PARTITION (partition_name) [USING index_name];`

- Recluster a table.
CLUSTER [VERBOSE];

Parameters

- **VERBOSE**
Enables the display of progress messages.
- **table_name**
Specifies the table name.
Value range: an existing table name
- **index_name**
Specifies the index name.
Value range: an existing index name
- **partition_name**
Specifies the partition name.
Value range: an existing partition name

Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create a partitioned table.
gaussdb=# CREATE TABLE tpcds.inventory_p1
(
    INV_DATE_SK          INTEGER          NOT NULL,
    INV_ITEM_SK          INTEGER          NOT NULL,
    INV_WAREHOUSE_SK    INTEGER          NOT NULL,
    INV_QUANTITY_ON_HAND INTEGER
)
DISTRIBUTE BY HASH(INV_ITEM_SK)
PARTITION BY RANGE(INV_DATE_SK)
(
    PARTITION P1 VALUES LESS THAN(2451179),
    PARTITION P2 VALUES LESS THAN(2451544),
    PARTITION P3 VALUES LESS THAN(2451910),
    PARTITION P4 VALUES LESS THAN(2452275),
    PARTITION P5 VALUES LESS THAN(2452640),
    PARTITION P6 VALUES LESS THAN(2453005),
    PARTITION P7 VALUES LESS THAN(MAXVALUE)
);

-- Create an index named ds_inventory_p1_index1.
gaussdb=# CREATE INDEX ds_inventory_p1_index1 ON tpcds.inventory_p1 (INV_ITEM_SK) LOCAL;

-- Cluster the tpcds.inventory_p1 table.
gaussdb=# CLUSTER tpcds.inventory_p1 USING ds_inventory_p1_index1;

-- Cluster the p3 partition.
gaussdb=# CLUSTER tpcds.inventory_p1 PARTITION (p3) USING ds_inventory_p1_index1;

-- Cluster the tables that can be clustered in the database.
gaussdb=# CLUSTER;

-- Delete the index.
gaussdb=# DROP INDEX tpcds.ds_inventory_p1_index1;

-- Drop the partitioned table.
gaussdb=# DROP TABLE tpcds.inventory_p1;
```

```
-- Drop the schema.  
gaussdb=# DROP SCHEMA tpccds CASCADE;
```

7.14.43 COMMENT

Description

Defines or changes the comment of an object.

Precautions

- Each object stores only one comment. Therefore, you need to modify a comment and issue a new **COMMENT** command to the same object. To delete the comment, write **NULL** at the position of the text string. When an object is deleted, the comment is automatically deleted.
- Currently, there is no security protection for viewing comments. Any user connected to a database can view all the comments for objects in the database. For shared objects such as databases, roles, and tablespaces, comments are stored globally so any user connected to any database in the cluster can see all the comments for shared objects. Therefore, do not put security-critical information in comments.
- To comment objects, you must be an object owner or user granted the **COMMENT** permission. System administrators have this permission by default.
- Roles do not have owners, so the rule for **COMMENT ON ROLE** is that you must be an administrator to comment on an administrator role, or have the **CREATEROLE** permission to comment on non-administrator roles. A system administrator can comment on all objects.

Syntax

```
COMMENT ON  
{  
  AGGREGATE agg_name (agg_type [, ...] ) |  
  CAST (source_type AS target_type) |  
  COLLATION object_name |  
  COLUMN { table_name.column_name | view_name.column_name } |  
  CONSTRAINT constraint_name ON table_name |  
  CONVERSION object_name |  
  DATABASE object_name |  
  DOMAIN object_name |  
  EXTENSION object_name |  
  FOREIGN DATA WRAPPER object_name |  
  
  FUNCTION function_name ( [ [ argname ] [ argmode ] argtype] [, ...] ) |  
  INDEX object_name |  
  LARGE OBJECT large_object_oid |  
  OPERATOR operator_name (left_type, right_type) |  
  OPERATOR CLASS object_name USING index_method |  
  OPERATOR FAMILY object_name USING index_method |  
  [ PROCEDURAL ] LANGUAGE object_name |  
  ROLE object_name |  
  SCHEMA object_name |  
  SERVER object_name |  
  TABLE object_name |  
  TABLESPACE object_name |  
  TEXT SEARCH CONFIGURATION object_name |  
  TEXT SEARCH DICTIONARY object_name |  
  TEXT SEARCH PARSER object_name |  
  TEXT SEARCH TEMPLATE object_name |  
  TYPE object_name |
```

```
VIEW object_name |  
TRIGGER trigger_name ON table_name  
}  
IS 'text';
```

Parameters

- **agg_name**
Specifies the new name of an aggregate function.
- **agg_type**
Specifies the data type of the aggregate function parameters.
- **source_type**
Specifies the source data type of the cast.
- **target_type**
Specifies the target data type of the cast.
- **object_name**
Specifies the name of an object.
- **table_name.column_name**
view_name.column_name
Specifies a column name. You can add the table name or view name as the prefix.
- **constraint_name**
Specifies the name of a table constraint.
- **table_name**
Specifies the name of a table.
- **function_name**
Specifies a function name.
- **argmode,argname,argtype**
Specifies the schema, name, and type of the function parameters.
- **large_object_oid**
Specifies the OID of a large object.
- **operator_name**
Specifies the name of the operator.
- **left_type,right_type**
Specifies the data type of the operator parameters (optionally schema-qualified). If the prefix or suffix operator does not exist, the **NONE** option can be added.
- **text**
Specifies the comment content.

Examples

```
-- Create a schema.  
gaussdb=# CREATE SCHEMA tpcds;  
  
-- Create the tpcds.customer table.  
gaussdb=# CREATE TABLE tpcds.customer
```

```
(
c_customer_sk      INTEGER      NOT NULL,
c_customer_id     CHAR(16)     NOT NULL
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.customer VALUES (50, 'AAAAAAAAABAAAAAA'),(100,
'AAAAAAAAACAAAAAA'),(150, 'AAAAAADAAAAAA');

gaussdb=# CREATE TABLE tpcds.customer_demographics_t2.
(
  CD_DEMO_SK      INTEGER      NOT NULL,
  CD_GENDER      CHAR(1)      ,
  CD_MARITAL_STATUS CHAR(1)    ,
  CD_EDUCATION_STATUS CHAR(20) ,
  CD_PURCHASE_ESTIMATE INTEGER  ,
  CD_CREDIT_RATING CHAR(10)   ,
  CD_DEP_COUNT   INTEGER      ,
  CD_DEP_EMPLOYED_COUNT INTEGER ,
  CD_DEP_COLLEGE_COUNT INTEGER
)
DISTRIBUTE BY HASH (CD_DEMO_SK);

-- Comment out the tpcds.customer_demographics_t2.cd_demo_sk column.
gaussdb=# COMMENT ON COLUMN tpcds.customer_demographics_t2.cd_demo_sk IS 'Primary key of
customer demographics table.';

-- Create a view consisting of rows with c_customer_sk less than 150.
gaussdb=# CREATE VIEW tpcds.customer_details_view_v2 AS
SELECT *
FROM tpcds.customer
WHERE c_customer_sk < 150;

-- Comment out the tpcds.customer_details_view_v2 view.
gaussdb=# COMMENT ON VIEW tpcds.customer_details_view_v2 IS 'View of customer detail';

-- Delete the view.
gaussdb=# DROP VIEW tpcds.customer_details_view_v2;

-- Delete tpcds.customer_demographics_t2.
gaussdb=# DROP TABLE tpcds.customer_demographics_t2;

-- Delete the tpcds.customer table.
gaussdb=# DROP TABLE tpcds.customer;

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

7.14.44 COMMIT | END

Description

COMMIT or END commits all operations of a transaction.

Precautions

Only the creator of a transaction or a system administrator can run the **COMMIT** command. The creation and commit operations must be in different sessions.

Syntax

```
{ COMMIT | END } [ WORK | TRANSACTION ] ;
```

Parameters

- **COMMIT | END**
Commits the current transaction and makes all changes made by the transaction become visible to others.
- **WORK | TRANSACTION**
Specifies an optional keyword, which has no effect except increasing readability.

Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create a table.
gaussdb=# CREATE TABLE tpcds.customer_demographics_t2
(
  CD_DEMO_SK          INTEGER          NOT NULL,
  CD_GENDER           CHAR(1)          ,
  CD_MARITAL_STATUS   CHAR(1)          ,
  CD_EDUCATION_STATUS CHAR(20)         ,
  CD_PURCHASE_ESTIMATE INTEGER          ,
  CD_CREDIT_RATING    CHAR(10)         ,
  CD_DEP_COUNT        INTEGER          ,
  CD_DEP_EMPLOYED_COUNT INTEGER         ,
  CD_DEP_COLLEGE_COUNT INTEGER         ,
)
DISTRIBUTE BY HASH (CD_DEMO_SK);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO tpcds.customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200,
'GOOD', 1, 0, 0);
gaussdb=# INSERT INTO tpcds.customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300, 'BAD',
1, 0, 0);

-- Commit the transaction to make all changes permanent.
gaussdb=# COMMIT;

-- Query data.
gaussdb=# SELECT * FROM tpcds.customer_demographics_t2;

-- Delete the tpcds.customer_demographics_t2 table.
gaussdb=# DROP TABLE tpcds.customer_demographics_t2;

-- Drop the schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[ROLLBACK](#)

7.14.45 COMMIT PREPARED

Description

Commits a prepared two-phase transaction. This function is for internal use only. You are advised not to use it.

Precautions

- The function is only available in maintenance mode (when GUC parameter **xc_maintenance_mode** is **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the transaction creators or system administrators can run the **COMMIT PREPARED** command. The creation and commit operations must be in different sessions.
- The transaction function is maintained automatically by the database, and should be not visible to users.

Syntax

```
COMMIT PREPARED transaction_id [ WITH commit_sequence_number ];
```

Parameters

- **transaction_id**
Specifies the identifier of the transaction to be committed. The identifier must be different from those for current prepared transactions.
- **commit_sequence_number**
Specifies the sequence number of the transaction to be committed. It is a 64-bit, incremental, unsigned number.

Examples

```
-- Start.
gaussdb=# BEGIN;

-- Prepare a transaction whose identifier is trans_test.
gaussdb=# PREPARE TRANSACTION 'trans_test';

-- Create a table.
gaussdb=# CREATE TABLE item1(id int);

--Commit the transaction whose identifier is trans_test.
gaussdb=# COMMIT PREPARED 'trans_test';

-- Delete the table.
gaussdb=# DROP TABLE item1;
```

Helpful Links

[PREPARE TRANSACTION](#) and [ROLLBACK PREPARED](#)

7.14.46 COPY

Description

Copies data between tables and files.

COPY FROM copies data from a file to a table, and **COPY TO** copies data from a table to a file.

Precautions

- When the **enable_copy_server_files** parameter is disabled, only the initial user is allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement. When the **enable_copy_server_files** parameter is enabled, users with the SYSADMIN permission or users who inherit permissions of the built-in role `gs_role_copy_files` are allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement. By default, database configuration files and key files are not allowed, and you can run **COPY FROM FILENAME** or **COPY TO FILENAME** for certificate files and audit logs to prevent unauthorized users from viewing or modifying sensitive files. When **enable_copy_server_files** is enabled, the administrator can use the GUC parameter **safe_data_path** to set the path for common users to import and export to the subpath of the set path. If this GUC parameter is not set (by default), the path used by common users is not blocked. This parameter reports an error for the relative path in the path of the COPY statement.
- COPY applies only to tables but not views.
- COPY TO requires the SELECT permission on the table to be read, and COPY FROM requires the INSERT permission on the table to be inserted.
- If a list of columns is specified, COPY copies only the data of the specified columns between the file and the table. If a table has any columns that are not in the column list, COPY FROM inserts default values for those columns.
- If a data source file is specified, the server must be able to access the file. If **STDIN** is specified, data flows between the client and the server. When entering data, use the **TAB** key to separate the columns of the table and use a backslash and a period (\.) in a new row to indicate the end of the input.
- COPY FROM throws an error if any row in the data file contains more or fewer columns than expected.
- The end of the data can be represented by a line that contains only backslashes and periods (\.). If data is read from a file, the end flag is unnecessary. If data is copied between client applications, an end tag must be provided.
- In COPY FROM, **\N** is an empty string. To enter the actual value **\N**, use **\\N**.
- COPY FROM can preprocess data using column expressions, but column expressions do not support subqueries.
- When a data format error occurs during COPY FROM execution, the transaction is rolled back. However, the error information is insufficient, making it difficult to locate the error data from a large amount of raw data.
- COPY FROM and COPY TO apply to low concurrency and local import and export of a small amount of data.
- When COPY is used in binary format, transcoding in distributed mode is not supported.
- **COPY** is a server command and its execution environment is the same as that of the database server process. **\COPY** is a client meta-command and its execution environment is the same as that of `gsql` on the client. Note that when the database and `gsql` are used in the sandbox environment, the **COPY** and **\COPY** commands both use the paths in the sandbox. When the database is used in the sandbox environment and `gsql` is used outside the sandbox, the **COPY** command uses the path inside the sandbox, and the **\COPY** command uses the path outside the sandbox.

- During the export using COPY TO, if the column data in the table contains the '\0' character, the column data will be truncated during the export. Only the data before '\0' will be exported.
- During the import using COPY FROM, if transcoding is not required, the size of a single row of data (including tuple metadata) is less than 1 GB minus 1 byte; if transcoding is required, the size of a single row of data is less than 256 MB minus 1 byte. In the following transcoding scenarios, special processing is required: The size for transcoding from UTF-8 to GB18030/GB18030_2022 must be less than 512 MB minus 1 byte, and that for transcoding from UTF-8 to GBK must be less than 1 GB minus 1 byte. If the data volume in a single row is too large and the value of **max_process_memory** is too small, an error message is displayed, indicating that the memory is insufficient. In this case, you need to adjust the value of **max_process_memory** and try again.

Syntax

- Copy data from a file to a table.

```
COPY table_name [ ( column_name [, ...] ) ]
FROM { 'filename' | STDIN }
[ [ USING ] DELIMITERS 'delimiters' ]
[ WITHOUT ESCAPING ]
[ LOG ERRORS ]
[ LOG ERRORS DATA ]
[ REJECT LIMIT 'limit' ]
[ [ WITH ] ( option [, ...] ) ]
| copy_option
| [ TRANSFORM ( { column_name [ data_type ] [ AS transform_expr ] }, ... ) ]
| [ FIXED FORMATTER ( { column_name( offset, length ) }, ... ) [ ( option [, ...] ) | copy_option
[ ...] ] ];
```

NOTE

The fixed formatter syntax is compatible with copy_option but incompatible with option. copy_option is incompatible with option. transform is compatible with copy_option and fixed formatter.

- Copy data from a table to a file.

```
COPY table_name [ ( column_name [, ...] ) ]
TO { 'filename' | STDOUT }
[ [ USING ] DELIMITERS 'delimiters' ]
[ WITHOUT ESCAPING ]
[ [ WITH ] ( option [, ...] ) ]
| copy_option
| [ FIXED FORMATTER ( { column_name( offset, length ) }, ... ) [ ( option [, ...] ) | copy_option
[ ...] ] ];
```

```
COPY query {(SELECT) | (VALUES)}
TO { 'filename' | STDOUT }
[ WITHOUT ESCAPING ]
[ [ WITH ] ( option [, ...] ) ]
| copy_option
| [ FIXED FORMATTER ( { column_name( offset, length ) }, ... ) [ ( option [, ...] ) | copy_option
[ ...] ] ];
```

NOTE

- The syntax constraints of COPY TO are as follows:
(query) is incompatible with [USING] DELIMITERS. If the data comes from a query result, COPY TO cannot specify [USING] DELIMITERS.
- Use spaces to separate copy_option following FIXED FORMATTER.
- **copy_option** is the native parameter, while **option** is the parameter imported by a compatible foreign table.

The syntax of the optional parameter **option** is as follows:

```
FORMAT 'format_name'  
| FORMAT binary  
| DELIMITER 'delimiter_character'  
| NULL 'null_string'  
| HEADER [ boolean ]  
| USEEOF [ boolean ]  
| FILEHEADER 'header_file_string'  
| FREEZE [ boolean ]  
| QUOTE 'quote_character'  
| ESCAPE 'escape_character'  
| EOL 'newline_character'  
| NOESCAPING [ boolean ]  
| FORCE_QUOTE { ( column_name [, ...] ) | * }  
| FORCE_NOT_NULL ( column_name [, ...] )  
| ENCODING 'encoding_name'  
| IGNORE_EXTRA_DATA [ boolean ]  
| FILL_MISSING_FIELDS [ boolean ]  
| COMPATIBLE_ILLEGAL_CHARS [ boolean ]  
| DATE_FORMAT 'date_format_string'  
| TIME_FORMAT 'time_format_string'  
| TIMESTAMP_FORMAT 'timestamp_format_string'  
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

The syntax of the optional parameter **copy_option** is as follows:

```
| NULL 'null_string'  
| HEADER  
| USEEOF  
| FILEHEADER 'header_file_string'  
| FREEZE  
| FORCE_NOT_NULL column_name [, ...]  
| FORCE_QUOTE { column_name [, ...] | * }  
| BINARY  
| CSV  
| QUOTE [ AS ] 'quote_character'  
| ESCAPE [ AS ] 'escape_character'  
| EOL 'newline_character'  
| ENCODING 'encoding_name'  
| IGNORE_EXTRA_DATA  
| FILL_MISSING_FIELDS  
| COMPATIBLE_ILLEGAL_CHARS  
| DATE_FORMAT 'date_format_string'  
| TIME_FORMAT 'time_format_string'  
| TIMESTAMP_FORMAT 'timestamp_format_string'  
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

Parameters

- **query**
Specifies that the results are to be copied.
Value range: Only one SELECT or VALUES command is supported. A semicolon (;) is not required at the end of the command.
- **table_name**
Specifies the name (possibly schema-qualified) of an existing table.
Value range: an existing table name
- **column_name**
Specifies an optional list of columns to be copied.
Value range: any columns. All columns will be copied if no column list is specified.
- **STDIN**
Specifies that input comes from the standard input.

- **STDOUT**
Specifies that output goes to the standard output.
- **FIXED**
Fixes column length. When the column length is fixed, **DELIMITER**, **NULL**, and **CSV** cannot be specified. When **FIXED** is specified, **BINARY**, **CSV**, and **TEXT** cannot be specified by **option** or **copy_option**.

 **NOTE**

The definition of fixed length is as follows:

- The column length of each record is the same.
 - Spaces are used for column padding. Columns of the numeric type are left-aligned and columns of the string type are right-aligned.
 - No delimiters are used between columns.
- **[USING] DELIMITERS 'delimiters'**

String that separates columns within each row (line) of the file. It cannot be larger than 10 bytes.

Value range: The delimiter in text format cannot include any of the following characters: \.abcdefghijklmnopqrstuvwxyz0123456789, but has no restriction for the CSV format.

Value range: The default value is a tab character in text format and a comma in CSV format.

 **NOTE**

Both **DELIMITER** and **DELIMITERS** can specify separators. However, **DELIMITERS** can be followed by syntax with bracket, but **DELIMITER** cannot be directly followed by bracket. Otherwise, a syntax error is reported.

- **WITHOUT ESCAPING**
Specifies, in the **TEXT** format, whether to escape the backslash (\) and its following characters.
Value range: text only
- **LOG ERRORS**
If this parameter is specified, the error tolerance mechanism for data type errors in the **COPY FROM** statement is enabled. Row errors are recorded in the **public.pgxc_copy_error_log** table in the database for future reference.
Value range: a value set while data is imported using **COPY FROM**.

 **NOTE**

The restrictions of this error tolerance parameter are as follows:

- This error tolerance mechanism captures only the data type errors (**DATA_EXCEPTION**) that occur during data parsing of **COPY FROM** on a CN. Other errors, such as network errors between CNs and DNs or expression conversion errors on DNs, are not captured.
 - Before enabling error tolerance for **COPY FROM** for the first time in a database, check whether the **public.pgxc_copy_error_log** table exists. If it does not, call the `copy_error_log_create()` function to create it. If it does, copy its data elsewhere, delete it, and call the `copy_error_log_create()` function to create the table. For details about columns in the **public.pgxc_copy_error_log** table, see [Table 7-87](#).
 - While a **COPY FROM** statement with specified **LOG ERRORS** is being executed, if **public.pgxc_copy_error_log** does not exist or does not have the table definitions compliant with those predefined in `copy_error_log_create()`, an error will be reported. Ensure that the error table is created using the `copy_error_log_create()` function. Otherwise, **COPY FROM** statements with error tolerance may fail to be run.
 - If existing error tolerance parameters (for example, **IGNORE_EXTRA_DATA**) of the **COPY** statement are enabled, the error of the corresponding type will be processed as specified by the parameters and no error will be reported. Therefore, the error table does not contain such error data.
- **LOG ERRORS DATA**
The differences between **LOG ERRORS DATA** and **LOG ERRORS** are as follows:
 - a. **LOG ERRORS DATA** fills the **rawrecord** column in the error tolerance table.
 - b. Only users with the super permission can use the parameter options of **LOG ERRORS DATA**.

NOTICE

- If error content is too complex, it may fail to be written to the error tolerance table by using **LOG ERRORS DATA**, causing the task failure.
 - For errors that cannot be read in certain code, the error codes are **ERRCODE_CHARACTER_NOT_IN_REPERTOIRE** and **ERRCODE_UNTRANSLATABLE_CHARACTER**. The **rawrecord** column is not recorded.
 - Files in binary format are not supported.
-
- **REJECT LIMIT 'limit'**
Used with the **LOG ERROR** parameter to set the upper limit of the tolerated errors in the **COPY FROM** statement. If the number of errors exceeds the limit, later errors will be reported based on the original mechanism.
Value range: a positive integer (1 to 2147483647) or **unlimited**
Default value: If **LOG ERRORS** is not specified, an error will be reported. If **LOG ERRORS** is specified, the default value is **0**.

 **NOTE**

Different from the GDS error tolerance mechanism, in the error tolerance mechanism described in **LOG ERRORS**, the count of **REJECT LIMIT** is calculated based on the number of data parsing errors on the CN where the **COPY FROM** statement is run, not based on the number of errors on each DN.

- **FORMATTER**

Defines the place of each column in the data file in fixed length mode.
Defines the place of each column in the data file in the **column(offset,length)** format.

Value range:

- The value of **offset** must be larger than 0. The unit is byte.
- The value of **length** must be larger than 0. The unit is byte.

The total length of all columns must be less than 1 GB.

Replace columns that are not in the file with null.

- **OPTION { option_name ' value ' }**

Specifies all types of parameters of a compatible foreign table.

- **FORMAT**

Specifies the format of the source data file in the foreign table.

Value range: **CSV**, **TEXT**, **FIXED**, and **BINARY**

- The CSV file can process newline characters efficiently, but cannot process certain special characters well.
- The TEXT file can process certain special characters efficiently, but cannot process newline characters well.
- In FIXED files, the column length of each record is the same. Spaces are used for padding, and the excessive part will be truncated.
- All data in the BINARY file is stored/read as binary format rather than as text. It is faster than the text and CSV formats, but a binary-format file is less portable.

Default value: **TEXT**

- **DELIMITER**

Specifies the character that separates columns within each row (line) of the file.

 NOTE

- The value of **delimiter** cannot be `\r` or `\n`.
- A delimiter cannot be the same as the null value. The delimiter for the CSV format cannot be same as the **quote** value.
- The delimiter for the TEXT format data cannot contain lowercase letters, digits, or special characters (`.``\`).
- The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
- You are advised to use multi-character delimiters or invisible delimiters. For example, you can use multi-characters (such as `$^&`) and invisible characters (such as `0x07`, `0x08`, and `0x1b`).
- To use a tab to isolate CSV data, set **delimiter** to `E't'`.

Value range: a multi-character delimiter within 10 bytes

Default value:

- A tab character in text format
- A comma (`,`) in CSV format
- No delimiter in FIXED format

- NULL

Specifies the string that represents a null value.

Value range:

- A null value cannot be `\r` or `\n`. The maximum length is 100 characters.
- A null value cannot be the same as the **delimiter** or **quote** value.

Default value:

- The default value for the CSV format is an empty string without quotation marks.
- The default value for the TEXT format is `\N`.

- HEADER

Specifies whether a file contains a header with the names of each column in the file. **header** is available only for CSV and FIXED files.

When data is imported, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

When data is exported, if header is **on**, **fileheader** must be specified. If **header** is **off**, an exported file does not contain a header.

Value range: **true/on** and **false/off**.

Default value: **false**

- USEEOF

The system does not report an error for `\"` in the imported data.

Value range: **true/on** and **false/off**.

Default value: **false**

- QUOTE

Specifies a quoted character string for a CSV file.

Default value: double quotation marks (""")

 NOTE

- The value of **quote** cannot be the same as that of the **delimiter** or **null** parameter.
- The value of **quote** must be a single-byte character.
- You are advised to set **quote** to an invisible character, such as **0x07**, **0x08**, or **0x1b**.

- ESCAPE

Specifies an escape character for a CSV file. The value must be a single-byte character.

Default value: double quotation mark ("""). If the value is the same as that of **quote**, it will be replaced by '\0'.

- EOL 'newline_character'

Specifies the newline character style of the imported or exported data file.

Value range: multi-character newline characters within 10 bytes.

Common newline characters include \r (0x0D), \n (0x0A), and \r\n (0x0D0A). Special newline characters include \$ and #.

 NOTE

- The EOL parameter supports only the TEXT format for data import and export and does not support the CSV or FIXED format for data import. For forward compatibility, the EOL parameter can be set to **0x0D** or **0x0D0A** for data export in the CSV or FIXED format.
- The value of **EOL** cannot be the same as that of the **delimiter** or **null** parameter.
- The EOL parameter value cannot contain the following characters: .abcdefghijklmnopqrstuvwxyz0123456789.

- FORCE_QUOTE { (column_name [, ...]) | * }

In **CSV COPY TO** mode, forces quotation marks to be used for all non-null values in each specified column. Null values are not quoted.

Value range: an existing column name

- FORCE_NOT_NULL (column_name [, ...])

Assigns a value to a specified column in **CSV COPY FROM** mode.

Value range: an existing column name

- ENCODING

Specifies the encoding format of a data file. The default value is the current database encoding format.

NOTICE

When COPY FROM is executed, **ENCODING** in the control file must be the same as the encoding format of the file. Otherwise, an error is reported or garbled characters are displayed in the imported data.

- IGNORE_EXTRA_DATA
Specifies whether to ignore excessive columns when the number of data source files exceeds the number of foreign table columns. This parameter is used only during data import.
Value range: **true/on** or **false/off**
 - **true/on**: If the number of columns in a data source file is greater than that defined by the foreign table, the extra columns at the end of a row are ignored.
 - **false/off**: If the number of columns in a data source file is greater than that defined by the foreign table, the following error message is reported:
extra data after last expected columnDefault value: **false**

NOTICE

If a newline character at the end of a row is missing and the row and another row are integrated into one, data in another row is ignored after the parameter is set to **true**.

- COMPATIBLE_ILLEGAL_CHARS
Specifies whether to tolerate invalid characters during data import. The parameter is valid only for data import using **COPY FROM**.
Value range: **true/on** and **false/off**.
 - **true/on**: No error message is reported and data import is not interrupted when there are invalid characters. Invalid characters are converted into valid ones, and then imported to the database.
 - **false/off**: An error occurs when there are invalid characters, and the import stops.Default value: **false/off**

NOTE

The rules for converting invalid characters are as follows:

1. '\0' is converted to a space.
 2. Other invalid characters are converted to question marks.
 3. When **compatible_illegal_chars** is set to **true/on**, after invalid characters such as **NULL**, **DELIMITER**, **QUOTE**, and **ESCAPE** are converted to spaces or question marks, an error message stating "illegal chars conversion may confuse COPY escape 0x20" will be displayed to remind you of possible parameter confusion caused by the conversion.
- FILL_MISSING_FIELDS
Specifies how to handle the problem that the last column of a row in a source data file is lost during data import.
Value range: **true/on** and **false/off**.
Default value: **false/off**

- DATE_FORMAT
Specifies the DATE format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.
Value range: a valid DATE value. For details, see [Date and Time Processing Functions and Operators](#).

 **NOTE**
If Oracle is specified as the compatible database, the DATE format is TIMESTAMP. For details, see [timestamp_format](#) below.
- TIME_FORMAT
Specifies the TIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.
Value range: a valid TIME value. Time zones cannot be used. For details, see [Date and Time Processing Functions and Operators](#).
- TIMESTAMP_FORMAT
Specifies the TIMESTAMP format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.
Value range: a valid TIMESTAMP value. Time zones cannot be used. For details, see [Date and Time Processing Functions and Operators](#).
- SMALLDATETIME_FORMAT
Specifies the SMALLDATETIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.
Value range: a valid SMALLDATETIME value. For details, see [Date and Time Processing Functions and Operators](#).
- **COPY_OPTION { option_name ' value ' }**
Specifies all types of native parameters of COPY.
 - NULL null_string
Specifies the string that represents a null value.

NOTICE

When using **COPY FROM**, any data item that matches this string will be stored as a null value, so make sure that you use the same string as you used with **COPY TO**.

Value range:

- A null value cannot be `\r` or `\n`. The maximum length is 100 characters.

- A null value cannot be the same as the **delimiter** or **quote** value.
Default value:
 - The default value for the TEXT format is `\N`.
 - The default value for the CSV format is an empty string without quotation marks.
- **HEADER**
Specifies whether a file contains a header with the names of each column in the file. **header** is available only for CSV and FIXED files.
When data is imported, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.
When data is exported, if header is **on**, **fileheader** must be specified. If **header** is **off**, an exported file does not contain a header.
- **USEEOF**
The system does not report an error for "\" in the imported data.
- **FILEHEADER**
Specifies a file that defines the content in the header for exported data. The file contains data description of each column.

NOTICE

- This parameter is available only when **header** is **on** or **true**.
- **fileheader** specifies an absolute path.
- The file can contain only one row of header information, and ends with a newline character. Excess rows will be discarded. (Header information cannot contain newline characters.)
- The length of the file including the newline character cannot exceed 1 MB.

-
- **FREEZE**
Sets the **COPY** loaded data row as **frozen**, like these data have executed **VACUUM FREEZE**.
This is a performance option of initial data loading. The data will be frozen only when the following three requirements are met:
 - The table being loaded has been created or truncated in the same transaction before copying.
 - There are no cursors open in the current transaction.
 - There are no original snapshots in the current transaction.

NOTE

When **COPY** is completed, all the other sessions will see the data immediately. However, this violates the general principle of MVCC visibility, and users should understand that this may cause potential risks.

- FORCE NOT NULL column_name [, ...]
Assigns a value to a specified column in **CSV COPY FROM** mode.
Value range: an existing column
- FORCE QUOTE { column_name [, ...] | * }
In **CSV COPY TO** mode, forces quotation marks to be used for all non-null values in each specified column. Null values are not quoted.
Value range: an existing column name
- BINARY
Specifies that data is stored and read in binary mode instead of text mode. In binary mode, you cannot declare **DELIMITER**, **NULL**, or **CSV**. When **BINARY** is specified, **CSV**, **FIXED**, and **TEXT** cannot be specified through **option** or **copy_option**.
- CSV
Enables the CSV mode. When **CSV** is specified, **BINARY**, **FIXED**, and **TEXT** cannot be specified through **option** or **copy_option**.
- QUOTE [AS] 'quote_character'
Specifies a quoted character string for a CSV file.
Default value: double quotation mark ("").

 NOTE

- The value of **quote** cannot be the same as that of the **delimiter** or **null** parameter.
 - The value of **quote** must be a single-byte character.
 - You are advised to set **quote** to an invisible character, such as **0x07**, **0x08**, or **0x1b**.
- ESCAPE [AS] 'escape_character'
Specifies an escape character for a CSV file. The value must be a single-byte character.
Default value: quotation mark ("). If the value is the same as that of **quote**, it will be replaced by '\0'.
 - EOL 'newline_character'
Specifies the newline character style of the imported or exported data file.
Value range: multi-character newline characters within 10 bytes.
Common newline characters include **\r** (0x0D), **\n** (0x0A), and **\r\n** (0x0D0A). Special newline characters include **\$** and **#**.

 NOTE

- The **EOL** parameter supports only the TEXT format for data import and export and does not support the CSV or FIXED format. For forward compatibility, the **EOL** parameter can be set to **0x0D** or **0x0D0A** for data export in the CSV or FIXED format.
 - The value of **EOL** cannot be the same as that of the **delimiter** or **null** parameter.
 - The EOL parameter value cannot contain the following characters: .abcdefghijklmnopqrstuvwxy0123456789.
- ENCODING 'encoding_name'

Specifies the name of a file encoding format.

Value range: a valid encoding format

Default value: current encoding format

NOTICE

When COPY FROM is executed, **ENCODING** in the control file must be the same as the encoding format of the file. Otherwise, an error is reported or garbled characters are displayed in the imported data.

- **IGNORE_EXTRA_DATA**

Specifies that when the number of data source files exceeds the number of foreign table columns, excess columns at the end of the row are ignored. This parameter is used only during data import.

If this parameter is not used and the number of columns in the data source file is greater than that defined in the foreign table, the following error information is displayed:

extra data after last expected column

- **COMPATIBLE_ILLEGAL_CHARS**

Specifies that invalid characters are tolerated during data import. Invalid characters are converted and then imported to the database. No error is reported and the import is not interrupted. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

If this parameter is not used, an error is reported when invalid characters are encountered during the import, and the import is interrupted.

NOTE

The rules for converting invalid characters are as follows:

1. '\0' is converted to a space.

2. Other invalid characters are converted to question marks.

3. When **compatible_illegal_chars** is set to **true/on**, after invalid characters such as **NULL**, **DELIMITER**, **QUOTE**, and **ESCAPE** are converted to spaces or question marks, an error message stating "illegal chars conversion may confuse COPY escape 0x20" will be displayed to remind you of possible parameter confusion caused by the conversion.

- **FILL_MISSING_FIELDS**

Specifies how to handle the problem that the last column of a row in a source data file is lost during data import.

Value range: **true/on** and **false/off**.

Default value: **false** or **off**

NOTICE

Do not specify this option. Currently, it does not enable error tolerance, but will make the parser ignore the said errors during data parsing on the CN. Such errors will not be recorded in the **COPY** error table (enabled using **LOG ERRORS REJECT LIMIT**) but will be reported later by DNs. Therefore, do not specify this option.

- **DATE_FORMAT** 'date_format_string'
Specifies the DATE format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.
Value range: a valid DATE value For details, see [Date and Time Processing Functions and Operators](#).

NOTE

If Oracle is specified as the compatible database, the DATE format is **TIMESTAMP**. For details, see **timestamp_format** below.

- **TIME_FORMAT** 'time_format_string'
Specifies the TIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.
Value range: a valid TIME value. Time zones cannot be used. For details, see [Date and Time Processing Functions and Operators](#).
- **TIMESTAMP_FORMAT** 'timestamp_format_string'
Specifies the **TIMESTAMP** format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.
Value range: a valid **TIMESTAMP** value. Time zones cannot be used. For details, see [Date and Time Processing Functions and Operators](#).
- **SMALLDATETIME_FORMAT** 'smalldatetime_format_string'
Specifies the **SMALLDATETIME** format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.
Value range: a valid **SMALLDATETIME** value. For details, see [Date and Time Processing Functions and Operators](#).
- **TRANSFORM** ({ column_name [data_type] [AS transform_expr] } [, ...])
Specify the conversion expression of each column in the table. **data_type** specifies the data type of the column in the expression parameter. **transform_expr** is the target expression that returns the result value whose data type is the same as that of the target column in the table. For details about the expression, see [Expressions](#).

NOTE

COPY FROM does not support expression conversion for distribution keys.

The following special backslash sequences are recognized by COPY FROM:

- **\b**: backslash (ASCII 8)
- **\f**: form feed (ASCII 12)
- **\n**: newline character (ASCII 10)
- **\r**: carriage return character (ASCII 13)
- **\t**: tab (ASCII 9)
- **\v**: vertical tab (ASCII 11)
- **\digits**: backslash followed by one to three octal digits specifies that the ASCII value is the character with that numeric code.
- **\xdigits**: backslash followed by an x and one or two hex digits specifies the character with that numeric code.

Permission Control Examples

```
gaussdb=> copy t1 from '/home/xy/t1.csv';
ERROR: COPY to or from a file is prohibited for security concerns
HINT: Anyone can COPY to stdout or from stdin. gsql's \copy command also works for anyone.
gaussdb=> grant gs_role_copy_files to xxx;
```

This error occurs because a non-initial user does not have the COPY permission. To solve this problem, enable the **enable_copy_server_files** parameter. Then, the administrator can use the COPY function, and common users need to join the **gs_role_copy_files** group.

Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.ship_mode table.
gaussdb=# CREATE TABLE tpcds.ship_mode
(
    SM_SHIP_MODE_SK      INTEGER      NOT NULL,
    SM_SHIP_MODE_ID     CHAR(16)     NOT NULL,
    SM_TYPE              CHAR(30)
    SM_CODE              CHAR(10)
    SM_CARRIER          CHAR(20)
    SM_CONTRACT         CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);

-- Insert a single data record into the tpcds.ship_mode table.
gaussdb=# INSERT INTO tpcds.ship_mode VALUES (1,'a','b','c','d','e');

-- Copy data from tpcds.ship_mode to the /home/omm/ds_ship_mode.dat file.
gaussdb=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode.dat';

-- Output tpcds.ship_mode to STDOUT.
gaussdb=# COPY tpcds.ship_mode TO STDOUT;

-- Output the data of tpcds.ship_mode to STDOUT. The parameters are as follows: The delimiter is ','
(delimiter',') and the encoding format is UTF8 (encoding'utf8').
gaussdb=# COPY tpcds.ship_mode TO STDOUT WITH (delimiter ',', encoding 'utf8');

-- Output the data of tpcds.ship_mode to STDOUT. The parameters are as follows: The import format is
CSV (format'CSV'), and the exported content of the SM_SHIP_MODE_SK column is enclosed in quotation
marks (force_quote(SM_SHIP_MODE_SK)).
gaussdb=# COPY tpcds.ship_mode TO STDOUT WITH (format 'CSV', force_quote(SM_SHIP_MODE_SK));

-- Create the tpcds.ship_mode_t1 table.
gaussdb=# CREATE TABLE tpcds.ship_mode_t1
```

```
(
  SM_SHIP_MODE_SK      INTEGER      NOT NULL,
  SM_SHIP_MODE_ID     CHAR(16)      NOT NULL,
  SM_TYPE              CHAR(30)
  SM_CODE              CHAR(10)
  SM_CARRIER          CHAR(20)
  SM_CONTRACT          CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);

-- Copy data from STDIN to the tpcds.ship_mode_t1 table.
gaussdb=# COPY tpcds.ship_mode_t1 FROM STDIN;

-- Enter a row of data as an example.
gaussdb=# COPY tpcds.ship_mode_t1 FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1 a a a a a
>> \.
Note: Enter a tab between data.

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table.
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat';

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, convert the
data using the TRANSFORM expression, and insert the 10 characters on the left of the SM_TYPE column
into the table.
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' TRANSFORM (SM_TYPE AS
LEFT(SM_TYPE, 10));

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, with the
import format set to TEXT (format 'text'), the delimiter set to \t (delimiter E'\t'), excessive columns
ignored (ignore_extra_data 'true'), and characters not escaped (noescaping 'true').
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' WITH(format 'text', delimiter
E'\t', ignore_extra_data 'true', noescaping 'true');

-- Copy data from the /home/omm/ds_ship_mode_fixed.dat file to the tpcds.ship_mode_t1 table, with the
import format set to FIXED, fixed-length format specified (FORMATTER(SM_SHIP_MODE_SK(0, 2),
SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20),
SM_CONTRACT(82,20))), excessive columns ignored (ignore_extra_data), and headers included (header).
gaussdb=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode_fixed.dat' FIXED
FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10),
SM_CARRIER(61,20), SM_CONTRACT(82,20)) header;
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode_fixed.dat' FIXED
FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10),
SM_CARRIER(61,20), SM_CONTRACT(82,20)) header ignore_extra_data;

-- Delete tables and the schema.
gaussdb=# DROP TABLE tpcds.ship_mode;
gaussdb=# DROP TABLE tpcds.ship_mode_t1;
gaussdb=# DROP SCHEMA tpcds;
```

7.14.47 CREATE AUDIT POLICY

Description

Creates a unified audit policy.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

The masking policy takes effect only after the security policy is enabled, that is, **enable_security_policy** is set to **on**.

NOTICE

When you use database links to perform operations on remote objects, the client initiates a database link request. The actual sender is the server, and the attributes such as the IP address of the sender are the values of the server. For details, see [DATABASE LINK](#).

Syntax

```
CREATE AUDIT POLICY [ IF NOT EXISTS ] policy_name { privilege_audit_clause | access_audit_clause } [, ... ]  
[ filter_group_clause ] [ ENABLE | DISABLE ];
```

- **privilege_audit_clause**
PRIVILEGES { DDL | ALL } [ON LABEL (resource_label_name [, ...])]
- **access_audit_clause**
ACCESS { DML | ALL } [ON LABEL (resource_label_name [, ...])]
- **filter_group_clause**
FILTER ON { filter_type (filter_value [, ...]) } [, ...]

Parameters

- **policy_name**
Specifies the audit policy name, which must be unique.
Value range: a string. It must comply with the [naming convention](#).
- **resource_label_name**
Specifies the resource label name.
- **DDL**
Specifies the operations that are audited in the database: **CREATE**, **ALTER**, **DROP**, **ANALYZE**, **COMMENT**, **GRANT**, **REVOKE**, **SET**, and **SHOW**.
If this parameter is set to **ANALYZE**, both ANALYZE and VACCUUM operations are audited.
- **DML**
Specifies the operations that are audited within the database: **SELECT**, **COPY**, **DEALLOCATE**, **DELETE**, **EXECUTE**, **INSERT**, **PREPARE**, **REINDEX**, **TRUNCATE**, and **UPDATE**.
- **ALL**
Specifies all DDL or DML operations supported in the database. When the form is { DDL | ALL }, **ALL** indicates all DDL operations. When the form is { DML | ALL }, **ALL** indicates all DML operations.
- **filter_type**
Specifies the types of information to be filtered by the policy, including **APP**, **ROLES**, and **IP**.
- **filter_value**
Indicates the detailed information to be filtered.
- **ENABLE|DISABLE**
Enables or disables the unified audit policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

```
-- Create users dev_audit and bob_audit.
gaussdb=# CREATE USER dev_audit PASSWORD '*****';
gaussdb=# CREATE USER bob_audit PASSWORD '*****';

-- Create table tb_for_audit.
gaussdb=# CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);

-- Create a resource label.
gaussdb=# CREATE RESOURCE LABEL adt_lb0 ADD TABLE(tb_for_audit);

-- Perform the CREATE operation on the database to create an audit policy.
gaussdb=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

-- Perform the SELECT operation on the database to create an audit policy.
gaussdb=# CREATE AUDIT POLICY adt2 ACCESS SELECT;

-- Create an audit policy to audit only the CREATE operations performed on the adt_lb0 resource by users dev_audit and bob_audit.
gaussdb=# CREATE AUDIT POLICY adt3 PRIVILEGES CREATE ON LABEL(adt_lb0) FILTER ON ROLES(dev_audit, bob_audit);

-- Create an audit policy to audit only the SELECT, INSERT, and DELETE operations performed on the adt_lb0 resource by users dev_audit and bob_audit using client tool gsql on the servers whose IP addresses are 10.20.30.40 and 127.0.0.0/24.
gaussdb=# CREATE AUDIT POLICY adt4 ACCESS SELECT ON LABEL(adt_lb0), INSERT ON LABEL(adt_lb0), DELETE FILTER ON ROLES(dev_audit, bob_audit), APP(gsql), IP('10.20.30.40', '127.0.0.0/24');

-- Delete the audit policy.
gaussdb=# DROP AUDIT POLICY adt1, adt2, adt3, adt4;

-- Delete the resource label.
gaussdb=# DROP RESOURCE LABEL adt_lb0;

-- Delete the tb_for_audit table.
gaussdb=# DROP TABLE tb_for_audit;

-- Delete the dev_audit and bob_audit users.
gaussdb=# DROP USER dev_audit, bob_audit;
```

Helpful Links

[ALTER AUDIT POLICY](#) and [DROP AUDIT POLICY](#)

7.14.48 CREATE BARRIER

Description

Creates a barrier for cluster nodes. The barrier can be used for data restoration. This function is for internal use only. You are advised not to use it.

Precautions

Generally, CREATE BARRIER is used only for backup and restoration. Therefore, CREATE BARRIER can be executed only in the following scenarios:

- The database initial user can run this command.
- If the backup and restoration mode is enabled on the CN, that is, the GUC parameter **operation_mode** is set to **on**, users with the OPRADMIN permission can run this command.

Syntax

```
CREATE BARRIER 'barrier_name';
```

Parameters

- **barrier_name**
(Required) Specifies the name of a barrier.
Value range: a string. It must comply with the [naming convention](#).

Examples

```
-- Specify the barrier name.  
gaussdb=# CREATE BARRIER 'barrier1';
```

7.14.49 CREATE CLIENT MASTER KEY

Description

The encrypted equality query feature adopts a multi-level encryption model. The master key encrypts the column key, and the column key encrypts data. This syntax is used to create a master key object.

Precautions

- This syntax is specific to a fully-encrypted database.
- When connecting to the database, you need to enable the connection parameters of the encrypted equality query feature on the database driver side before running this syntax.
- The master key is provided by an external key manager. This syntax processes only information such as the key source and key ID. The following external key managers are supported:
 - a. Huawei Cloud KMS huawei_kms
- Before using this syntax, set environment variables for the external key manager on the database driver side. For details, see section "Setting Encrypted Equality Query" in *Feature Guide*.

Syntax

```
CREATE CLIENT MASTER KEY client_master_key_name WITH (KEY_STORE = key_store_name, KEY_PATH = "key_path_value", ALGORITHM = algorithm_type);
```

Parameters

- **client_master_key_name**
This parameter is used as the name of a key object. In the same namespace, the value of this parameter must be unique.
Value range: a string. It must comply with the [naming convention](#).
- **KEY_STORE**
External key manager. For details about the value, see [Table 7-161](#).
- **KEY_PATH**
Each key is managed by an external key manager and the key path format varies depending on the key manager. The value is a character string. For

details, see [Table 7-161](#). A character string is enclosed in single or double quotation marks. If the length of a character string exceeds 64 characters, only single quotation marks can be used.

- **ALGORITHM**

Encryption algorithm used by the key. For details about the value, see [Table 7-161](#).

Table 7-161 Parameter values for different key managers

KEY_STORE	KEY_PATH	ALGORITHM
huawei_kms	Format: '{KmsApiUrl}/{Key ID}' Reference: 'https://kms.{Project}.myhuaweicloud.com/v1.0/{Project ID}/kms/{Key ID}' Example: 'https://kms.cn-north-4.myhuaweicloud.com/v1.0/00000000000000000000000000000000/kms/00000000-0000-0000-0000-000000000000'	AES_256

Example (Using gsql to Connect to the Database Server)

Before using this syntax, you need to enable KMS and configure parameters for accessing KMS. For details, see "Setting Encrypted Equality Queries" in *Feature Guide*.

Helpful Links

[DROP CLIENT MASTER KEY](#)

7.14.50 CREATE COLUMN ENCRYPTION KEY

Description

Creates a CEK that can be used to encrypt a specified column in a table.

Precautions

This syntax is specific to a fully-encrypted database.

When using **gsql** to connect to a database server, you need to use the **-C** parameter to enable the fully-encrypted database.

The CEK object created using this syntax can be used for column-level encryption. When defining a column in a table, you can specify a CEK object to encrypt the column.

Syntax

```
CREATE COLUMN ENCRYPTION KEY column_encryption_key_name WITH VALUES(CLIENT_MASTER_KEY = client_master_key_name, ALGORITHM = algorithm_type, ENCRYPTED_VALUE = encrypted_value);
```

Parameters

- **column_encryption_key_name**
Key object name. In the same namespace, the name must be unique.
Value range: a string. It must comply with the [naming convention](#).
- **CLIENT_MASTER_KEY**
Specifies the CMK used to encrypt the CEK. The value is the CMK object name, which is created using the CREATE CLIENT MASTER KEY syntax.
- **ALGORITHM**
Encryption algorithm to be used by the CEK. The value can be **AEAD_AES_256_CBC_HMAC_SHA256**, **AEAD_AES_128_CBC_HMAC_SHA256**, **AEAD_AES_256_CTR_HMAC_SHA256**, **AES_256_GCM**, or **SM4_SM3**.
The data expansion rates of different encryption algorithms are sorted as follows: **AEAD_AES_256_CTR_HMAC_SHA256** < **AES_256_GCM** < **AEAD_AES_256_CBC_HMAC_SHA256** = **AEAD_AES_128_CBC_HMAC_SHA256** = **SM4_SM3**. The **AEAD_AES_256_CTR_HMAC_SHA256** and **AES_256_GCM** encryption algorithms are recommended.
- **ENCRYPTED_VALUE (optional)**
Specifies the key password defined by the user. The key password contains 28 to 256 characters. The security strength of a key containing 28 characters complies with AES-128. If AES-256 is used, the key password must contain 39 characters. If this parameter is not specified, a 256-character key is automatically generated.

NOTICE

- SM algorithm constraints: SM2, SM3, and SM4 are Chinese national cryptography standards. To avoid legal risks, these algorithms must be used together. If you specify the SM4 algorithm to encrypt CEKs when creating a CMK, you must specify the SM3 and SM4 algorithms (**SM4_SM3**) to encrypt data when creating CEKs.
 - Constraints on the **ENCRYPTED_VALUE** column: If the CMK generated by Huawei KMS is used to encrypt the CEK and the **ENCRYPTED_VALUE** column is used to transfer the key in the **CREATE COLUMN ENCRYPTION KEY** syntax, the length of the input key must be an integer multiple of 16 bytes.
-

Example (Using gsql to Connect to the Database Server)

Before using this syntax, you need to enable KMS and configure parameters for accessing KMS. For details, see "Setting Encrypted Equality Queries" in *Feature Guide*.

Helpful Links

[ALTER COLUMN ENCRYPTION KEY](#) and [DROP COLUMN ENCRYPTION KEY](#)

7.14.51 CREATE CONVERSION

Description

Defines a new conversion between two character set encodings. This function is for internal use only. You are advised not to use it.

Precautions

- The **DEFAULT** parameter indicates that the conversion between the source encoding and the target encoding is executed by default between the client and the server. To support this usage, bidirectional conversion must be defined, that is, both conversion from A to B and conversion from B to A are supported.
- To perform conversion, you must have the EXECUTE permission on function and the CREATE permission on the target schema.
- SQL_ASCII cannot be used for either source encoding or target encoding because the server behavior is hardwired when SQL_ASCII "encoding" is involved.
- You can remove user-defined conversions using DROP CONVERSION.

Syntax

```
CREATE [ DEFAULT ] CONVERSION name  
FOR 'source_encoding' TO 'dest_encoding' FROM function_name
```

Parameters

- **DEFAULT**
Specifies that the conversion is the default conversion from the source encoding to the target encoding. There should be only one default conversion for each encoding pair in a schema.
- **name**
Specifies the name of the conversion, which can be restricted by the schema. If not restricted by a schema, the conversion is defined in the current schema. The conversion name must be unique in a schema.
- **source_encoding**
Source encoding name.
- **dest_encoding**
Target encoding name.
- **function_name**
Function used for conversion. A function name can be restricted by a schema. If not, the function is found in the path.

The function must be in the following format:

```
conv_proc(  
integer, -- Source encoding ID  
integer, -- Target encoding ID
```

```
cstring, -- Source character string (C character string ending with a null value)
internal,-- Target (filled with a null-terminated C character string)
integer -- Length of the source string
) RETURNS void;
```

CAUTION

Currently, only internal creation is supported.

7.14.52 CREATE DATABASE

Description

Creates a database. By default, the new database will be created only by cloning the standard system database **template0**.

Precautions

- A user who has the CREATEDB permission or a system administrator can create a database.
- It cannot be executed inside a transaction block.
- If an error message similar to "could not initialize database directory" is displayed during database creation, the possible cause is that the permission on the data directory in the file system is insufficient or the disk is full.

Syntax

```
CREATE DATABASE database_name
  [ [ WITH ] { [ OWNER [=] user_name ] |
  [ TEMPLATE [=] template ] |
  [ ENCODING [=] 'encoding' ] |
  [ LC_COLLATE [=] 'lc_collate' ] |
  [ LC_CTYPE [=] 'lc_ctype' ] |
  [ DBCOMPATIBILITY [=] 'compatibility_type' ] |
  [ TABLESPACE [=] tablespace_name ] |
  [ CONNECTION LIMIT [=] connlimit ] |
  [ DBTIMEZONE [=] 'time_zone' ] }[...] ];
```

Parameters

- **database_name**
Specifies the database name.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **OWNER [=] user_name**
Specifies the owner of the new database. If omitted, the default owner is the current user.
Value range: an existing username.
- **TEMPLATE [=] template**
Specifies a template name. That is, the template from which the database is created. GaussDB creates a database by copying data from a template database. GaussDB has two default template databases **template0** and **template1** and a default user database **postgres**.

Value range: **template0**.

- **ENCODING [=] 'encoding'**

Specifies the character encoding used by the database. The value can be a string (for example, **SQL_ASCII**) or an integer.

By default, the encoding format of the template database is used. By default, the codes of the template databases **template0** and **template1** are related to the OS environment. The character encoding of **template1** cannot be changed. To change the encoding, use **template0** to create a database.

Common values are **GBK**, **UTF8**, **Latin1**, and **GB18030**. The supported character sets are as follows:

Table 7-162 Supported character sets

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
BIG5	Big Five	Traditional Chinese	No	No	1-2	WIN950, Windows950
EUC_CN	Extended Unix Code-CN	Simplified Chinese	Yes	Yes	1-3	-
EUC_JP	Extended Unix Code-JP	Japanese	Yes	Yes	1-3	-
EUC_JIS_2004	Extended Unix Code-JP, JIS X 0213	Japanese	Yes	No	1-3	-
EUC_KR	Extended Unix Code-KR	Korean	Yes	Yes	1-3	-
EUC_TW	Extended Unix Code-Taiwan, China	Traditional Chinese	Yes	Yes	1-3	-
GB18030	National standards	Chinese	Yes	No	1-4	-
GB18030_2022	National standards	Chinese	Yes	No	1-4	-

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
GBK	Extended national standards	Simplified Chinese	Yes	No	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Latin/Cyrillic	Yes	Yes	1	-
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/Arabic	Yes	Yes	1	-
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/Greek	Yes	Yes	1	-
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/Hebrew	Yes	Yes	1	-
JOHAB	JOHAB	Korean	No	No	1-3	-
KOI8R	KOI8-R	Cyrillic (Russian)	Yes	Yes	1	KOI8
KOI8U	KOI8-U	Cyrillic (Ukrainian)	Yes	Yes	1	-
LATIN1	ISO 8859-1, ECMA 94	Western European languages	Yes	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European languages	Yes	Yes	1	ISO88592

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
LATIN3	ISO 8859-3, ECMA 94	South European languages	Yes	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European languages	Yes	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	Yes	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Germanic languages	Yes	Yes	1	ISO885910
LATIN7	ISO 8859-13	Baltic languages	Yes	Yes	1	ISO885913
LATIN8	ISO 8859-14	Celtic languages	Yes	Yes	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 with Euro and accents	Yes	Yes	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	Romanian	Yes	No	1	ISO885916
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	No	1-4	-
SJIS	Shift JIS	Japanese	No	No	1-2	Mskanji, ShiftJIS, WIN932, Windows932

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
SHIFT_JIS_2004	Shift JIS, JIS X 0213	Japanese	No	No	1-2	-
SQL_ASCII	Unspecified (see the text)	<i>Any</i>	Yes	No	1	-
UHC	Unified Hangul Code	Korean	No	No	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	All	Yes	Yes	1-4	Unicode
WIN866	Windows CP866	Cyrillic	Yes	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	No	1	-
WIN1250	Windows CP1250	Central European languages	Yes	Yes	1	-
WIN1251	Windows CP1251	Cyrillic	Yes	Yes	1	WIN
WIN1252	Windows CP1252	Western European languages	Yes	Yes	1	-
WIN1253	Windows CP1253	Greek	Yes	Yes	1	-
WIN1254	Windows CP1254	Turkish	Yes	Yes	1	-
WIN1255	Windows CP1255	Hebrew	Yes	Yes	1	-

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
WIN1256	Windows CP1256	Arabic	Yes	Yes	1	-
WIN1257	Windows CP1257	Baltic languages	Yes	Yes	1	-
WIN1258	Windows CP1258	Vietnamese	Yes	Yes	1	ABC, TCVN, TCVN5712, VSCII

 **CAUTION**

- Note that not all client APIs support the preceding character sets.
 - The SQL_ASCII setting is different from other settings. If the character set of the server is SQL_ASCII, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If this parameter is set to SQL_ASCII, no code conversion occurs. Therefore, this setting is not basically used to declare the specified encoding used, because this declaration ignores the encoding. In most cases, if you use any non-ASCII data, do not use the SQL_ASCII setting because the database will not be able to convert or verify non-ASCII characters.
-

NOTICE

- The character set encoding of the new database must be compatible with the local settings (**LC_COLLATE** and **LC_CTYPE**).
 - When the specified character encoding set is **GBK**, some uncommon Chinese characters cannot be directly used as object names. This is because the byte encoding overlaps with the ASCII characters `@A-Z[\]^_`a-z{}` when the second byte of the GBK ranges from 0x40 to 0x7E. `@[\]^_`{}` is an operator in the database. If it is directly used as an object name, a syntax error will be reported. For example, the GBK hexadecimal code is **0x8240**, and the second byte is **0x40**, which is the same as the ASCII character `@`. Therefore, the character cannot be used as an object name. If you need to use this function, you can add double quotation marks ("") to avoid this problem when creating and accessing objects.
 - If the client code is A and the server code is B, the conversion between encoding formats A and B must exist in the database. For details about encoding format conversion supported by the database, see the system catalog **PG_CONVERSION**. (If the encoding format cannot be converted, it is recommended that the encoding format on the client be the same as that on the server. You can change the encoding format on the client by setting the **client_encoding** parameter.)
 - If you want to set the database character set encoding to **GB18030_2022** and the client encoding to **GB18030**, ensure that the client OS supports **GB18030_2022**. If the **GB18030** character set versions are incompatible with each other, data inconsistency may occur. In addition, if historical data needs to be switched to the **GB18030_2022** database, follow the database switching process to migrate data.
-
- **LC_COLLATE [=] 'lc_collate'**
Specifies the character set used by the new database. For example, set this parameter by using **lc_collate = 'zh_CN.gbk'**.
The use of this parameter affects the sort order of strings (for example, the order of using **ORDER BY** for execution and the order of using indexes on text columns). By default, the sorting order of the template database is used.
Value range: character sets supported by the OS.
 - **LC_CTYPE [=] 'lc_ctype'**
Specifies the character class used by the new database. For example, set this parameter by using **lc_ctype = 'zh_CN.gbk'**. The use of this parameter affects the classification of characters, such as uppercase letters, lowercase letters, and digits. By default, the character classification of the template database is used.
Value range: character classes supported by the OS.

 **NOTE**

- The value ranges of **lc_collate** and **lc_ctype** depend on the character sets supported by the local environment. For example, in the Linux OS, you can run the **locale -a** command to obtain the list of character sets supported by the OS. When using the **lc_collate** and **lc_ctype** parameters, you can select the required character sets and character categories.
- If the specified character encoding set is **GB18030_2022**, the value ranges of **LC_COLLATE** and **LC_CTYPE** are the same as those of **GB18030**.

- **DBCOMPATIBILITY [=] 'compatibility_type'**

Specifies the compatible database type. The default value is **MySQL**.

Value range: **MYSQL**, **TD**, **ORA**, and **PG**. MySQL, Teradata, Oracle and PostgreSQL are compatible, respectively.

 **NOTE**

- For ORA compatibility, the database treats empty strings as **NULL** and replaces **DATE** with **TIMESTAMP(0) WITHOUT TIME ZONE**.
 - When a character string is converted to an integer, if the input is invalid, the input will be converted to 0 due to **MYSQL** compatibility, and an error will be reported due to other compatibility issues.
 - For PG compatibility, **CHAR** and **VARCHAR** are counted by character. For other compatibility types, they are counted by byte. For example, for the UTF-8 character set, **CHAR(3)** can store three Chinese characters in PG compatibility scenarios, but can store only one Chinese character in other compatibility scenarios.
 - If an error is reported during statement execution, A-format or B-format is displayed in some error information. A-format indicates ORA-format, and B-format indicates **MYSQL**-format.
- **TABLESPACE [=] tablespace_name**
Specifies the tablespace of the database.
Value range: an existing tablespace name.
 - **CONNECTION LIMIT [=] connlimit**
Specifies the maximum number of concurrent connections that can be made to the new database.

NOTICE

- System administrators are not restricted by this parameter.
- **connlimit** is calculated for each each CN. The number of connections in a cluster is calculated using the following formula: Number of connections in a cluster = **connlimit** x Number of normal CNs.

Value range: an integer in the range $[-1, 2^{31} - 1]$. The default value is **-1**, indicating that there is no limit.

The restrictions on character encoding are as follows:

- If the locale is set to **C** (or **POSIX**), all encoding types are allowed. For other locale settings, the character encoding must be the same as that of the locale.
- The encoding and region settings must match the template database, except that **template0** is used as a template. This is because other databases may contain data that does not match the specified encoding, or may contain indexes whose sorting order is affected by **LC_COLLATE** and **LC_CTYPE**. Copying this data will invalidate the indexes in the new database. **template0** does not contain any data or indexes that may be affected.
- **DBTIMEZONE [=] 'time_zone'**
Specifies the time zone of the new database. For example, you can set this parameter by setting **DBTIMEZONE** to **'+00:00'**. This parameter affects the time zone of the new database. The **PRC** is used by default.

Value range: name and abbreviation of the time zone supported by the OS, or the timestamp ranges from -15:59 to +15:00.

Examples

```
-- Create users jim and tom.
gaussdb=# CREATE USER jim PASSWORD '*****';
gaussdb=# CREATE USER tom PASSWORD '*****';

-- Create database music using GBK (the local encoding type is also GBK).
gaussdb=# CREATE DATABASE music ENCODING 'GBK' template = template0;

-- Create database music2 and specify user jim as its owner.
gaussdb=# CREATE DATABASE music2 OWNER jim;

-- Create database music3 using template template0 and specify user jim as its owner.
gaussdb=# CREATE DATABASE music3 OWNER jim TEMPLATE template0;

-- Set the maximum number of connections to database music to 10.
gaussdb=# ALTER DATABASE music CONNECTION LIMIT= 10;

-- Rename database music to music4.
gaussdb=# ALTER DATABASE music RENAME TO music4;

-- Change the owner of database music2 to user tom.
gaussdb=# ALTER DATABASE music2 OWNER TO tom;

-- Drop databases.
gaussdb=# DROP DATABASE music2;
gaussdb=# DROP DATABASE music3;
gaussdb=# DROP DATABASE music4;

-- Drop the jim and tom users.
gaussdb=# DROP USER jim;
gaussdb=# DROP USER tom;

-- Create a database compatible with the TD format.
gaussdb=# CREATE DATABASE td_compatible_db DBCOMPATIBILITY 'TD';

-- Create a database compatible with the ORA format.
gaussdb=# CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'ORA';

-- Drop the databases that are compatible with the TD and ORA formats.
gaussdb=# DROP DATABASE td_compatible_db;
gaussdb=# DROP DATABASE ora_compatible_db;
```

Helpful Links

[ALTER DATABASE](#) and [DROP DATABASE](#)

Suggestions

- **create database**
Database cannot be created in a transaction.
- **ENCODING**
If the new database Encoding does not match the template database (SQL_ASCII) ('GBK', 'UTF8', 'LATIN1', 'GB18030', or 'GB18030_2022'), **template [=] template0** must be specified.

7.14.53 CREATE DATABASE LINK

Description

Creates a database link object. For details about database links, see [DATABASE LINK](#).

Precautions

- Do not use database links to connect to the initial user.
- The initial user is not allowed to create, modify, or delete database links.
- If the upgrade is not committed, the database link cannot be created.
- If `CURRENT_USER` is used or the `CONNECT TO` connection string is omitted, the initial username and empty password of the current database are used for connection, resulting in connection failure.

Syntax

```
CREATE [ PUBLIC ] DATABASE LINK dblink  
[ CONNECT TO { CURRENT_USER | user IDENTIFIED BY password } ] [ USING ( option 'value' [...] ) ];
```

Parameters

- **PUBLIC**
Creates a public database link visible to all users. If this clause is omitted, the database link is private and available only to the current user.
- **dblink**
Indicates the name of the database link to be created.
- **user**
Indicates the name of the user for connecting to the remote end of the created database link.
- **password**
Indicates the password of the user for connecting to the remote end of the created database link.
- **CURRENT_USER**
Uses the initial username and empty password of the current database for connection.
- **USING (option 'value' [, ...])**
Specifies parameters such as the IP address, port number, and remote database name of the database to be connected. The supported options are as follows:
 - **host**: specifies the IP addresses to be connected. IPv6 addresses are not supported. Multiple IP addresses can be specified using character strings separated by commas (.). Currently, SSL settings, encrypted databases, and certificate authentication are not supported. If no IP address is specified, this parameter is left empty by default.
 - **port**: specifies the port number for connection. If this parameter is not specified, the default value **5432** is used.

- **dbname**: specifies the name of the database to be connected. If this parameter is not specified, the username used for connecting to the remote end is used by default.
- **fetch_size**: specifies the amount of data obtained from the remote end each time. The value of **fetch_size** ranges from 0 to 2147483647. The default value is **100**.

NOTICE

- You can write only part of the preceding options in the brackets after USING.
- If the keyword USING is not written, the content in the brackets is not written as well.
- When a database link is created, the system does not check whether the connection is successful. If related keywords are missing, an error may be reported.
- In a distributed system, CNs and DNs may be deployed on different servers. Therefore, you are advised not to set the **host** parameter to **127.0.0.1** or **localhost**. Otherwise, the connection may fail.

Example

```
-- Create a user with the SYSADMIN permission.
gaussdb=# CREATE USER user01 WITH SYSADMIN PASSWORD '*****';
gaussdb=# SET ROLE user01 PASSWORD '*****';

-- Create a private database link.
gaussdb=# CREATE DATABASE LINK private_dblink CONNECT TO 'user1' IDENTIFIED BY '*****' USING
(host '192.168.11.11',port '54399',dbname 'db01');

-- Delete the private database link.
gaussdb=# DROP DATABASE LINK private_dblink;

-- Create a public database link.
gaussdb=# CREATE PUBLIC DATABASE LINK public_dblink CONNECT TO 'user1' IDENTIFIED BY '*****'
USING (host '192.168.11.11',port '54399',dbname 'db01');

-- Delete the public database link.
gaussdb=# DROP PUBLIC DATABASE LINK public_dblink;

-- Delete the created user.
gaussdb=# RESET ROLE;
gaussdb=# DROP USER user01 CASCADE;
```

7.14.54 CREATE DIRECTORY

Function

CREATE DIRECTORY creates a directory. The directory defines an alias for a path in the server file system and is used to store data files used by users. Users can read and write these files through the **dbe_file** advanced package.

The read and write permissions for the directory can be granted to specified users to provide permission control for **dbe_file**.

Precautions

- When **enable_access_server_directory** is set to **off**, only the initial user is allowed to create directory objects. When **enable_access_server_directory** is set to **on**, the user with the SYSADMIN permission and the user who inherits the **gs_role_directory_create** permission of the built-in role can create directory objects.
- By default, the user who creates a directory has the read and write permissions on the directory.
- The default owner of a directory is the user who creates the directory.
- A directory cannot be created for the following paths:
 - The path contains special characters.
 - The path is a relative path.
- The following validity check is performed during directory creation:
 - Check whether the path exists in the OS. If it does not exist, a message is displayed, indicating the potential risks.
 - Check whether the database initial user **omm** has the read, write, and execution permissions on the created directory. If the user does not have all the permissions, a message is displayed, indicating the potential risks.
- In a cluster, ensure that the path is the same on all the nodes. Otherwise, the path may fail to be found on some nodes when the directory is used.

Syntax

```
CREATE [OR REPLACE] DIRECTORY directory_name  
AS 'path_name';
```

Parameter Description

- **directory_name**
Specifies name of a directory.
Value range: a string. It must comply with the naming convention.
- **path_name**
Specifies the OS path for which a directory is to be created.
Value range: a valid OS path

Examples

```
-- Create a directory.  
gaussdb=# CREATE OR REPLACE DIRECTORY dir AS '/tmp/';  
  
-- Delete the directory.  
gaussdb=# DROP DIRECTORY dir;
```

Helpful Links

[ALTER DIRECTORY](#) and [DROP DIRECTORY](#)

7.14.55 CREATE FOREIGN DATA WRAPPER

Description

Creates a foreign data wrapper. The user who created the foreign data wrapper becomes its owner.

Precautions

- The name of the foreign data wrapper must be unique in the database.
- Only initial users and system administrators can create foreign data wrappers.

Syntax

```
CREATE FOREIGN DATA WRAPPER name  
[ HANDLER handler_function | NO HANDLER ]  
[ VALIDATOR validator_function | NO VALIDATOR ]  
[ OPTIONS ( option 'value' [, ... ] ) ];
```

Parameters

- **name**
Specifies the name of the foreign data wrapper to be created.
- **HADNLER handler_function**
handler_function is the name of a registered function used to retrieve execution functions for foreign tables. The handler function must have no arguments, and its return type must be `fdw_handler`.
It is possible to create foreign data wrappers without processor functions, but foreign tables that use such wrappers can only be declared and cannot be accessed.
- **VALIDATOR validator_function**
validator_function is the name of a registered function used to check the common options provided to the foreign data wrapper, as well as the options for the foreign server, user mapping, and foreign table that use the foreign data wrapper. If there is no validator function or **NO VALIDATOR** is declared, the option is not checked during creation. (The foreign data wrapper may ignore or reject invalid option descriptions at runtime, depending on the implementation.) The validator function must accept two arguments: one is of `text[]` type, which will contain an array of options stored in the system directory, and the other is of `OID` type, which will be the `OID` of the system directory that contains the options. The return type is ignored. The function should report invalid options using the `ereport(ERROR)` function.
- **OPTIONS (option 'value' [, ...])**
This clause declares options for the new foreign data wrapper. The allowed option names and values are specific to each foreign data wrapper and are validated by the validator function of the foreign data wrapper. The option name must be unique.

Example

```
-- Create a useless foreign-data wrapper dummy.  
gaussdb=# CREATE FOREIGN DATA WRAPPER dummy;
```

```
-- Create a foreign data wrapper file with the handler function file_fdw_handler.  
gaussdb=# CREATE FOREIGN DATA WRAPPER file HANDLER file_fdw_handler;  
  
-- Create a foreign data wrapper mywrapper with some options.  
gaussdb=# CREATE FOREIGN DATA WRAPPER mywrapper OPTIONS (debug 'true');
```

Helpful Links

[ALTER FOREIGN DATA WRAPPER](#) and [DROP FOREIGN DATA WRAPPER](#)

7.14.56 CREATE FUNCTION

Description

Creates a function.

Precautions

- If the parameters or return values of a function have precision, the precision is not checked.
- When creating a function, you are advised to explicitly specify the schemas of tables in the function definition. Otherwise, the function may fail to be executed.
- **current_schema** and **search_path** specified by SET during function creation are invalid. **search_path** and **current_schema** before and after function execution should be the same.
- If a function has output parameters, the GUC parameter **set behavior_compat_options** must be set to '**proc_outparam_override**' for the output parameters to take effect. When the function is called using SELECT or CALL, an actual parameter must be provided in the position of the output parameter. Otherwise, the function fails to be called.
- After **REPLACE** is specified, a new function is created instead of replacing a function if the number of parameters, parameter type, or return value is different.
- You cannot create overloaded functions with different formal parameter names (the function name and parameter list type are the same).
- You cannot create a function that has the same name and parameter list as a stored procedure.
- Formal parameters cannot be overloaded if only the custom ref cursor type is different from the sys_refcursor type.
- Function overloading is not supported if only the returned data types are different.
- Function overloading is not supported if only the default values are different.
- When an overloaded function is called, the variable type must be specified.
- If an undeclared variable is used in a function, an error is reported when the function is called.
- You can use the SELECT statement to specify different parameters using identical functions.
- When you create a function, you cannot insert other agg functions out of the avg function or other functions.

- In common cluster mode, return values, parameters, and variables cannot be set to the tables of the node groups that are not installed in the system by default. The internal statements of SQL functions cannot be executed on such tables.
- By default, the permissions to execute new functions are granted to PUBLIC users. For details, see [GRANT](#). By default, a user inherits the permissions of the PUBLIC role. Therefore, the user has the permission to execute a function and view the definition of the function. In addition, to execute the function, the user must have the USAGE permission on the schema to which the function locates. You can revoke the default execution permissions from the PUBLIC role when creating a function and grant the function execute permission to users as needed. To avoid the time window during which new functions can be accessed by all users, create functions and set function execution permissions in a transaction. After the database object isolation attribute is enabled, common users can view only the definitions of functions that they have permission to execute.
- If a function is defined as **IMMUTABLE** or **SHIPPABLE**, avoid **INSERT**, **UPDATE**, **DELETE**, **MERGE**, and **DDL** operations in the function because the CN needs to determine the execution node for these operations. Otherwise, an error may occur. If DDL operations are performed on a function of the IMMUTABLE or SHIPPABLE type, database objects on each node may be inconsistent. To resolve this problem, create the VOLATILE PL/SQL function on the CN, run the EXECUTE statement in the function definition to dynamically execute the DDL operation for repairing system objects, and then use the EXECUTE DIRECT ON syntax to call the repair function on the specified DN.
- When functions without parameters are called inside another function, you can omit brackets and call functions using their names directly.
- If the parameter **behavior_compat_options** is not set to '**proc_outparam_override**', the **OUT** output parameter of the function is directly called by an anonymous block or stored procedure, and the return value is used as the first value of the **OUT** output parameter. As a result, the calling fails. To correctly use the **OUT** and **IN OUT** output parameters, set the parameter **behavior_compat_options** to '**proc_outparam_override**'. For details, see the example.
- When other functions with output parameters are called in a function and an assignment expression, set the GUC parameter **behavior_compat_options** to '**proc_outparam_override**', define variables of the same type as the output parameters in advance, and use the variables as output parameters to call other functions with output parameters for the output parameters to take effect. Otherwise, the output parameters of the called functions will be ignored.
- After the GUC parameter '**proc_outparam_override**' is set, if the return type of the function is setof, the output parameter will not take effect.
- Oracle-compatible functions support viewing, exporting, and importing parameter comments.
- Oracle-compatible functions support viewing, exporting, and importing comments between IS/AS and plsql_body.
- Users granted the CREATE ANY FUNCTION permission can create or replace functions in the user schemas.

- The default permission on a function is SECURITY INVOKER. To change the permission to SECURITY DEFINER, set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**.
- When a function is created, it depends on an undefined object. If **behavior_compat_options** is set to **'plpgsql_dependency'**, the creation can be executed and a warning message is displayed. If **behavior_compat_options** is not set to **'plpgsql_dependency'**, the creation cannot be executed.
- When **behavior_compat_options** is set to **'plpgsql_dependency'**, if function A is called in the function and function B is contained in the input and output parameters of function A, function B will not establish a dependency. For example, **functionA(functionB())**. **gs_dependencies** only creates dependency with function A.
- If a view directly depends on an O-style function and the **behavior_compat_options** parameter is set to **'plpgsql_dependency'**, the view can be accessed when the function is created again. However, if the **behavior_compat_options** parameter is not set to **'plpgsql_dependency'**, the view cannot be accessed.
- When creating a function, you cannot use the function itself as the default value of input parameter.
- The function with **OUT** parameter cannot be called by SQL statement.
- The function with **OUT** parameter cannot be called by SELECT INTO syntax.
- Functions with **OUT** parameters cannot be called in nested mode.

Example:

```
b := func(a,func(c,1));
```

Should be changed to:

```
tmp := func(c,1); b := func(a,tmp);
```

- When a function is created, the type of the return value of the function is not checked.
- If a function with the definer specified is created in a schema of another user, the function will be executed by another user, which may cause unauthorized operations. Therefore, exercise caution when performing this operation.
- In the schema of the O&M administrator, only the initial user and schema owner can create objects. The schema that does not allow other users to create and modify objects is the schema of the O&M administrator.
- If the **out** parameter is used as the output parameter in an expression, the expression does not take effect in the following scenarios: (a) The execute immediate sqlv using func syntax is used to execute a function. (b) The select func into syntax is used to execute a function. (c) DML statements such as INSERT and UPDATE are used to execute a function. (d) The select where a=func() statement is used. (e) When a function with the **out** output parameter is used as an input parameter, that is, **fun (func (out b), a)**, the **out b** parameter does not take effect.
- When a stored procedure with the out parameter is called, you can set the GUC parameter **behavior_compat_options** to **'proc_outparam_transfer_length'** to pass the parameter length. The specifications and constraints are as follows:

- a. The following types are supported: CHAR(n), CHARACTER(n), NCHAR(n), VARCHAR(n), VARYING(n), VARCHAR2(n), and NVARCHAR2(n).
- b. If the out parameter does not take effect (for example, **perform**), the length does not need to be transferred.
- c. The following types do not support precision transfer: NUMERIC, DECIMAL, NUMBER, FLOAT, DEC, INTEGER, TIME, TIMESTAMP, INTERVAL, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE, TIME WITHOUT TIME ZONE, and TIMESTAMP WITHOUT TIME ZONE.
- d. The parameter length can be transferred regardless of whether the GUC parameter **set behavior_compat_options** is set to **proc_outparam_override**.

Syntax

- Syntax (compatible with PostgreSQL) for creating a user-defined function:

```
CREATE [ OR REPLACE ] FUNCTION function_name
( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ] } [, ...] ] )
[ RETURNS rettype [ DETERMINISTIC ] | RETURNS TABLE ( { column_name column_type }
[, ...] ) ]
LANGUAGE lang_name
[
  {IMMUTABLE | STABLE | VOLATILE }
  | {SHIPPABLE | NOT SHIPPABLE}
  | WINDOW
  | [ NOT ] LEAKPROOF
  | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
  | [ {EXTERNAL } SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER |
AUTHID CURRENT_USER}
  | {fenced | not fenced}

  | COST execution_cost
  | ROWS result_rows
  | SET configuration_parameter { {TO | =} value | FROM CURRENT } }
][...]
{
  AS 'definition'
  | AS 'obj_file', 'link_symbol'
};
```

- ORA-compatible syntax for creating a user-defined function:

```
CREATE [ OR REPLACE ] FUNCTION function_name
( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ] } [, ...] ] )
RETURN rettype [ DETERMINISTIC ]
[
  {IMMUTABLE | STABLE | VOLATILE }
  | {SHIPPABLE | NOT SHIPPABLE}

  | {FENCED | NOT FENCED}
  | [ NOT ] LEAKPROOF
  | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
  | [ {EXTERNAL } SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER |
AUTHID DEFINER | AUTHID CURRENT_USER
}
]
| COST execution_cost
| ROWS result_rows
| SET configuration_parameter { {TO | =} value | FROM CURRENT
| LANGUAGE lang_name
][...]
{
  IS | AS
} plsql_body
/
```

Parameters

- **function_name**

Specifies the name of the function to be created (optionally schema-qualified).

Value range: a string. It must comply with the [naming convention](#). The value contains a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the function name.

- **argname**

Specifies the parameter name of the function.

Value range: a string. It must comply with the [naming convention](#). The value contains a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the function parameter name.

- **argmode**

Specifies the parameter mode of the function.

Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**. The default value is **IN**. Only the parameter of the **OUT** mode can be followed by **VARIADIC**. The parameters of **OUT** and **INOUT** cannot be used in the function definition of RETURNS TABLE.

 **NOTE**

VARIADIC specifies parameters of the array type.

- **argtype**

Specifies the data type of a function parameter. You can use **%ROWTYPE** to indirectly reference the type of a table. For details, see [Variable Definition Statements](#).

- **expression**

Specifies the default expression of a parameter.

 **NOTE**

- If **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, the default expression is not supported when the parameter is in INOUT mode.
- It is recommended that you define all default parameters after all non-default parameters.
- If a function with default parameters is called, input parameters are added to the function from left to right. If inputs of non-default parameters are missing, an error is reported.
- If **proc_uncheck_default_param** is enabled and a function with default parameters is called, input parameters are added to the function from left to right. If an input of a non-default parameter is missing, the previous default value is used to fill this parameter.
- When **a_format_version** is set to **10c**, **a_format_dev_version** is set to **s1**, **proc_outparam_override** is disabled, and the function parameters include the output parameter **out** and **default**, the default value cannot be used.

- **rettype**

Specifies the return data type. Same as **argtype**, **%TYPE** or **%ROWTYPE** can also be used to indirectly reference types.

When there is **OUT** or **IN OUT** parameter, the RETURNS clause can be omitted. If the clause is not omitted, the result type of the clause must be the same as that of the output parameter. The result type of the clause is the same as that of a single output parameter.

The SETOF modifier indicates that the function will return a set of items, rather than a single item.

- **column_name**
Specifies the column name.
- **column_type**
Specifies the column type.
- **definition**
Specifies a string constant defining a function. Its meaning depends on the language. It can be an internal function name, a path pointing to a target file, an SQL query, or text in a procedural language.
- **DETERMINISTIC**
Specifies an API compatible with the SQL syntax. You are advised not to use it.
- **LANGUAGE lang_name**
Specifies the name of the language that is used to implement the function. It can be **SQL**, **C**, **internal**, or the name of a customized process language. To ensure downward compatibility, the name can use single quotation marks. Contents in single quotation marks must be capitalized.
Due to compatibility issues, no matter which language is specified when an ORA-style database is created, the language used is plpgsql.

 **NOTE**

- When an internal function is defined, if AS specifies the function as an internal system function, the parameter type, number of parameters, and return value type of the new function must be the same as those of the internal system function, and the user who creates the internal function must have the permission to execute the internal system function.
- Only users with the SYSADMIN permission can create internal functions.
- **WINDOW**
Specifies that the function is a window function. This is currently only useful for functions written in C. The **WINDOW** attribute cannot be changed when replacing an existing function definition.

NOTICE

For a customized window function, the value of **LANGUAGE** can only be **internal**, and the referenced internal function must be a window function.

-
- **IMMUTABLE**
Specifies that the function always returns the same result if the parameter values are the same.
 - **STABLE**

Specifies that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.

- **VOLATILE**

Specifies that the function value can change in a single table scan and no optimization is performed.

- **SHIPPABLE**

NOT SHIPPABLE

Specifies whether the function can be pushed down to DNs for execution.

- Functions of the IMMUTABLE type can always be pushed down to DNs.
- Functions of the STABLE or VOLATILE type can be pushed down to DNs only if their attribute is **SHIPPABLE**.

NOTICE

The function or stored procedure of the SHIPPABLE/IMMUTABLE type cannot contain EXCEPTION or call functions or stored procedures that contain EXCEPTION.

- **LEAKPROOF**

Specifies that the function has no side effects. LEAKPROOF can be set only by system administrators.

- **CALLED ON NULL INPUT**

Declares that some parameters of the function can be called in normal mode if the parameter values are null. This parameter can be omitted.

- **RETURNS NULL ON NULL INPUT**

STRICT

Specifies that the function always returns null whenever any of its parameters is null. If this parameter is specified, the function is not executed when there are null parameters; instead a null result is returned automatically.

RETURNS NULL ON NULL INPUT and **STRICT** have the same functions.

- **EXTERNAL**

The purpose is to be compatible with SQL statements and it is optional. This feature applies to all functions, not only external functions.

- **SECURITY INVOKER**

AUTHID CURRENT_USER

Specifies that the function will be executed with the permissions of the user who calls it. This parameter can be omitted.

SECURITY INVOKER and **AUTHID CURRENT_USER** have the same functions.

- **SECURITY DEFINER**

AUTHID DEFINER

Specifies that the function will be executed with the permissions of the user who created it.

AUTHID DEFINER and **SECURITY DEFINER** have the same functions.

- **FENCED**
NOT FENCED
Specifies whether the function is executed in fenced mode or not fenced mode. In **NOT FENCED** mode, a function is executed in a CN or DN process. In **FENCED** mode, a function is executed in a new fork process, which does not affect CN or DN processes.
Application scenarios:
 - Develop or debug a function in **FENCED** mode and execute it in **NOT FENCED** mode. This reduces the overhead of the fork process and communication.
 - Perform complex OS operations, such as opening a file, and processing signals and threads in **FENCED** mode; otherwise, the GaussDB database execution may be affected.
 - Customize PL/SQL functions. If this parameter is not specified, the default value **NOT FENCED** is used and the **FENCED** execution mode is not supported.
- **COST execution_cost**
Estimates the execution cost of a function.
The unit of **execution_cost** is **cpu_operator_cost**.
Value range: ≥ 0
- **ROWS result_rows**
Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.
Value range: ≥ 0 . The default value is **1000**.
- **configuration_parameter**
 - **value**
Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** disables the setting.
Value range: a string
 - **DEFAULT**
 - **OFF**
 - **RESET**Specifies the default value.
 - **FROM CURRENT**
Uses the value of **configuration_parameter** of the current session.
- **plsql_body**
Specifies the PL/SQL stored procedure body.

NOTICE

When you perform password-related operations, such as user creation, password change, and encryption/decryption, in a function body, the password will be recorded in the system catalogs and logs in plaintext. To prevent sensitive information leakage, you are advised not to perform operations on the function body related to sensitive information, such as passwords or keys.

Examples

```
-- Create an ORA-compatible database.
gaussdb=# CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'ORA';
CREATE DATABASE
-- Switch to another database.
gaussdb=# \c ora_compatible_db
-- Define a function as SQL query.
gaussdb=# CREATE FUNCTION func_add_sql(integer, integer) RETURNS integer
  AS 'select $1 + $2;'
  LANGUAGE SQL
  IMMUTABLE
  RETURNS NULL ON NULL INPUT;

-- Add an integer by parameter name using PL/pgSQL.
gaussdb=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
  BEGIN
    RETURN i + 1;
  END;
$$ LANGUAGE plpgsql;

-- Return a record containing multiple output parameters.
gaussdb=# CREATE FUNCTION func_dup_sql(in int, out f1 int, out f2 text)
  AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
  LANGUAGE SQL;

gaussdb=# SELECT * FROM func_dup_sql(42);

-- Compute the sum of two integers and return the result (if the input is null, the returned result is null).
gaussdb=# CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

-- Alter the execution rule of function add to IMMUTABLE (that is, the same result is returned if the
parameter remains unchanged).
gaussdb=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) IMMUTABLE;

-- Alter the name of function add to add_two_number.
gaussdb=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) RENAME TO add_two_number;

-- Change the owner of function add to omm.
gaussdb=# ALTER FUNCTION add_two_number(INTEGER, INTEGER) OWNER TO omm;

-- Delete the function.
gaussdb=# DROP FUNCTION add_two_number;
gaussdb=# DROP FUNCTION func_increment_sql;
gaussdb=# DROP FUNCTION func_dup_sql;
gaussdb=# DROP FUNCTION func_increment_plsql;
gaussdb=# DROP FUNCTION func_add_sql;

-- If the parameter behavior_compat_options is not set to 'proc_outparam_override', the OUT output
parameter of the function is directly called by an anonymous block or stored procedure, and the return
```

```
value is used as the first value of the OUT output parameter. As a result, the call fails.
gaussdb=# CREATE TYPE rec AS(c1 int, c2 int);
gaussdb=# CREATE OR REPLACE FUNCTION func(a in out rec, b in out int) return int
AS
BEGIN
  a.c1:=100;
  a.c2:=200;
  b:=300;
  return 1;
END;
/
DECLARE
  r rec;
  b int;
BEGIN
  func(r,b); -- Not supported.
END;
/
ERROR: cannot assign non-composite value to a row variable
CONTEXT: PL/SQL function inline_code_block line 4 at SQL statement

gaussdb=# CREATE OR REPLACE FUNCTION func_001(a in out date, b in out date) --#add in & inout
#default value
RETURN integer
AS
BEGIN
  raise info '%', a;
  raise info '%', b;
RETURN 1;
END;
/
gaussdb=# DECLARE
  date1 date := '2022-02-02';
  date2 date := '2022-02-02';
BEGIN
  func_001(date1, date2);
END;
/
INFO: 2022-02-02 00:00:00
CONTEXT: PL/SQL function inline_code_block line 5 at SQL statement
INFO: 2022-02-02 00:00:00
CONTEXT: PL/SQL function inline_code_block line 5 at SQL statement
ERROR: invalid input syntax for type timestamp: "1"
CONTEXT: PL/SQL function inline_code_block line 5 at SQL statement

gaussdb=# CREATE OR REPLACE FUNCTION func_001(a in out INT, b in out date) --#add in & inout
#default value
RETURN INT
AS
BEGIN
  raise info '%', a;
  raise info '%', b;
RETURN a;
END;
/
gaussdb=# DECLARE
  date1 int := 1;
  date2 date := '2022-02-02';
BEGIN
  func_001(date1, date2);
END;
/
INFO: 1
CONTEXT: PL/SQL function inline_code_block line 5 at SQL statement
INFO: 2022-02-02 00:00:00
CONTEXT: PL/SQL function inline_code_block line 5 at SQL statement
ANONYMOUS BLOCK EXECUTE
```

Helpful Links

[ALTER FUNCTION](#) and [DROP FUNCTION](#)

Suggestions

- analyse | analyze
 - Do not run **ANALYZE** in a transaction or anonymous block.
 - Do not run **ANALYZE** in a function or stored procedure.

7.14.57 CREATE GLOBAL INDEX

Description

Creates a global secondary index (GSI) on a specified table.

The GSIs allow users to define indexes that are inconsistent with the distribution of base tables. In this way, single-node plans for querying non-distribution keys of base tables and unique/primary key constraints on non-distribution keys of base tables are achieved.

NOTE

1. GSI creation is not allowed during the upgrade uncommitted observation.
2. You are advised to create a NOT NULL constraint on the index key of at least one column of GSI to improve IUD performance.
3. You are advised to create ordinary indexes in the GSI base table to improve IUD execution efficiency.
4. The execution plan generated when a GSI is created is to pull the base table data back to the CN and then deliver the data to the DN where the GSI is located. Therefore, the performance of a large-scale cluster with a large amount of data is much worse than that of using an ordinary index. For example, it may take about one hour to create a GSI with tens of millions of data records on six SSDs. The GSI creation mechanism will be optimized in later versions and the current single point will be changed to parallel DNs to improve the creation efficiency.

Precautions

- Similar to the base table constraints, the distribution keys of GSIs do not support the UPDATE operation.
- The GSI can be created only in GTM-LITE mode. If a GSI is created in other GTM modes, an error is reported.
- Common UB-tree indexes can be created. Expression indexes and some other indexes are not supported. Astore does not support UB-tree indexes other than GSIs and does not support partition creation for GSIs.
- The GSI whose distribution is the same as that of the base table cannot be created. Otherwise, an error will be reported during the execution.
- Online GSI creation or rebuild is not supported and a syntax error will be reported. PARALLEL is not supported, and **parallel_workers** will be set to **0** on the DN.
- Hash-distributed GSIs can be created for row-store Astore tables and partitioned tables whose base tables are hash-distributed. The base tables

cannot be replicate, hash bucket, list/range distribution, or Ustore table. GSIs of these tables do not support distribution other than hash distribution.

- If the base table is a partitioned table, the GSI supports a maximum of 27 columns. If the base table is not a partitioned table, the GSI supports a maximum of 28 columns (including the index key and distribution key).
- UPSERT, IUD returning on base tables with GSIs, TABLE ACCESS BY INDEX ROWID, MERGE INTO, CLUSTER, and SQL PATCH are not supported.
- Operations that will invalidate the GSIs in the current version:
 - VACUUM FULL
 - VACUUM FULL on a single table: All GSIs on the table are invalidated.
 - VACUUM FULL on a database: All GSIs in the database are invalidated.
 - REINDEX
 - REINDEX on a single table (offline): All GSIs on the table are invalidated.
 - REINDEX on a database (offline): All GSIs in the database are invalidated.
 - CLUSTER
 - CLUSTER on a single table using an ordinary index: All GSIs on the table are invalidated.
 - CLUSTER on a single table: All GSIs on the table are invalidated.
 - CLUSTER on a database: All GSIs on the tables that have been clustered in the database are invalidated.
 - COPY/GDS
All GSIs on the table are invalidated.
 - PARTITION
MERGE PARTITION, EXCHANGE PARTITION, TRUNCATE PARTITION, DROP PARTITION and SPLIT PARTITION invalidate all GSIs on a partitioned table. EXCHANGE PARTITION invalidates all GSIs on an ordinary table.
- For batch scenarios such as INSERT INTO SELECT and UPDATE/DELETE, the execution plan goes back to the CN, and the performance is poor (similar to the performance of GSI creation).
- For INSERT, UPDATE, and DELETE, distributed execution plans are used, which may cause performance loss.
- If `_new$$` or `_NEW$$` is added to a column name or `ctid`, `xc_node_hash`, `xmin`, `xmax` or `tableoid` (when the base table is a partitioned table), a GSI fails to be created for the base table whose column names are repeated.
- If the VACUUM FULL, CLUSTER, or REINDEX operation on a table is interrupted, the GSI on the table may turn to the **UNUSABLE** state. In this case, an error will be reported when the GSI is queried. You are advised to run **REINDEX INDEX** to rebuild the GSI.

- When **ENABLE_PBE_OPTIMIZATION** is disabled, the GSI layer of the INSERT, UPDATE, and DELETE operations uses a gplan.

Syntax

```
CREATE GLOBAL [ UNIQUE ] INDEX [ [schema_name.]index_name ] ON table_name [ USING method ]  
( { column_name [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] }, ... )  
[CONTAINING (containing_colname)] [DISTRIBUTE BY hash(dist_colnames)]  
[ TABLESPACE tablespace_name ];
```

Parameters

- **UNIQUE**
Creates a unique index. In this way, the system checks whether new values are unique in the index column. If the inserted or updated value causes duplicate records, an error is reported.
- **schema_name**
Specifies the schema name.
Value range: an existing schema name
- **index_name**
Specifies the name of the index to be created. No schema name can be included here; the index is always created in the same schema as its parent table.
Value range: a string. It must comply with the [naming convention](#).
- **table_name**
Specifies the name of the table to be indexed (optionally schema-qualified).
Value range: an existing table name
- **USING method**
Specifies the name of the index method to be used.
Value range: UB-tree. The multi-version B-tree index is provided. The index page contains transaction information.
- **column_name**
Specifies the name of the column on which an index is to be created.
If the index mode supports multi-column indexes, multiple columns can be declared. A maximum of 28 columns can be declared for a non-partitioned base table, and a maximum of 27 columns can be declared for a partitioned base table.
- **COLLATE collation**
Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **select * from pg_collation** command to query collation rules from the **pg_collation** system catalog. The default collation rule is the row starting with **default** in the query result.
- **opclass**
Specifies the name of an operator class. An operator class can be specified for each column of an index. The operator class identifies the operators to be used by the index for that column.
- **ASC**

- Specifies an ascending (default) sort order.
- **DESC**
Specifies a descending sort order.
 - **NULLS FIRST**
Specifies that null values appear before non-null values in the sort ordering. This is the default when **DESC** is specified.
 - **NULLS LAST**
Specifies that null values appear after non-null values in the sort ordering. This is the default when **DESC** is not specified.
 - **CONTAINING**
Specifies the base table attributes other than index keys contained in the GSI.
 - **containing_colname**
Specifies the base table attributes other than the index keys contained in the GSI.
 - **DISTRIBUTE BY**
Specifies the distribution key of the GSI, which is different from that of the base table, and only the hash distribution can be used.
 - **dist_colnames**
Specifies the distribution key of the GSI.
It must be contained in **column_name**.
 - **TABLESPACE tablespace_name**
Specifies the tablespace for an index. If no tablespace is specified, the default tablespace is used.
Value range: an existing tablespace name.

Examples

```
-- Create a base table test that contains three columns.
gaussdb=# CREATE TABLE test(c1 int, c2 int, c3 int);

-- Create a GSI on the c2 column of the test table containing the c3 column based on the hash distribution
of the c2 column.
gaussdb=# CREATE GLOBAL INDEX idx_gsi_1 ON test(c2) CONTAINING(c3) DISTRIBUTE BY HASH(c2);

-- Create a base table test2 that contains three columns.
gaussdb=# CREATE TABLE test2(c1 int, c2 int, c3 int);

-- Create a GSI on the c2 column of the test2 table containing the c3 column based on the hash
distribution of the c2 column.
gaussdb=# CREATE GLOBAL INDEX idx_gsi_2 ON test2(c2) CONTAINING(c3) ;

-- Create a base table test3 that contains three columns.
gaussdb=# CREATE TABLE test3(c1 int, c2 int, c3 int);

-- Create a GSI in UNIQUE form on the c2 column of the test3 table, which is based on the hash
distribution of the c2 column by default.
gaussdb=# CREATE GLOBAL UNIQUE INDEX idx_gsi_3 ON test3(c2) DISTRIBUTE BY HASH(c2);

-- Drop the index.
gaussdb=# DROP INDEX idx_gsi_1;
gaussdb=# DROP INDEX idx_gsi_2;
gaussdb=# DROP INDEX idx_gsi_3;

-- Drop the table.
```

```
gaussdb=# DROP TABLE test1;  
gaussdb=# DROP TABLE test2;  
gaussdb=# DROP TABLE test3;
```

Helpful Links

[ALTER INDEX](#), [CREATE INDEX](#), and [DROP INDEX](#)

7.14.58 CREATE GROUP

Description

Creates a user group.

Precautions

CREATE GROUP is an alias for CREATE ROLE, and it is not a standard SQL syntax and not recommended. Users can use CREATE ROLE directly.

Syntax

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ]  
[ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' [ EXPIRED ] | DISABLE };
```

The syntax of the optional action clause is as follows:

```
{SYSADMIN | NOSYSADMIN}  
| {MONADMIN | NOMONADMIN}  
| {OPRADMIN | NOOPRADMIN}  
| {POLADMIN | NOPOLADMIN}  
| {AUDITADMIN | NOAUDITADMIN}  
| {CREATEDB | NOCREATEDB}  
| {USEFT | NOUSEFT}  
| {CREATEROLE | NOCREATEROLE}  
| {INHERIT | NOINHERIT}  
| {LOGIN | NOLOGIN}  
| {REPLICATION | NOREPLICATION}  
| {VCADMIN | NOVCADMIN}  
| {PERSISTENCE | NOPERSISTENCE}  
| CONNECTION LIMIT connlimit  
| VALID BEGIN 'timestamp'  
| VALID UNTIL 'timestamp'  
  
| USER GROUP 'groupuser'  
| PERM SPACE 'spacelimit'  
| NODE GROUP logic_group_name  
| IN ROLE role_name [, ...]  
| IN GROUP role_name [, ...]  
| ROLE role_name [, ...]  
| ADMIN role_name [, ...]  
| USER role_name [, ...]  
| SYSID uid  
| DEFAULT TABLESPACE tablespace_name  
| PROFILE DEFAULT  
| PROFILE profile_name  
| PGUSER
```

Parameters

See [Parameters](#) in section "CREATE ROLE."

Examples

```
-- Create group my_group.  
gaussdb=# CREATE GROUP my_group PASSWORD '*****';  
  
-- Delete the group.  
gaussdb=# DROP GROUP my_group;
```

Helpful Links

[ALTER GROUP](#), [DROP GROUP](#), and [CREATE ROLE](#)

7.14.59 CREATE INCREMENTAL MATERIALIZED VIEW

Description

CREATE INCREMENTAL MATERIALIZED VIEW creates a fast-refresh materialized view, and you can refresh the data of the materialized view by using REFRESH MATERIALIZED VIEW (complete-refresh) and REFRESH INCREMENTAL MATERIALIZED VIEW (fast-refresh).

CREATE INCREMENTAL MATERIALIZED VIEW is similar to CREATE TABLE AS, but it remembers the query used to initialize the view, so it can refresh data later. A materialized view has many attributes that are the same as those of a table, but does not support temporary materialized views.

Precautions

- Fast-refresh materialized views cannot be created on database link tables, temporary tables, or global temporary tables.
- Fast-refresh materialized views support only simple filter queries and UNION ALL queries of base tables.
- Fast-refresh materialized views do not support the WITH, GROUP BY, ORDER BY, LIMIT, and WINDOW clauses, the DISTINCT and AGG operators, and subqueries except UNION ALL.
- The distribution key cannot be specified when a fast-refresh materialized view is created.
- After a fast-refresh materialized view is created, most DDL operations in the base table are no longer supported.
- The IUD operation cannot be performed on fast-refresh materialized views.
- After a fast-refresh materialized view is created, you need to run the **REFRESH** command to synchronize the materialized view with the base table when the base table data changes.
- Ustore does not support the creation and use of materialized views.

Syntax

```
CREATE INCREMENTAL MATERIALIZED VIEW mv_name  
  [ (column_name [, ...] ) ]  
  [ TABLESPACE tablespace_name ]  
  AS query;
```

Parameters

- **mv_name**
Name (optionally schema-qualified) of the materialized view to be created.
Value range: a string. It must comply with the [naming convention](#).
- **column_name**
Column name in the new materialized view. The materialized view supports specified columns. The number of specified columns must be the same as the number of columns in the result of the subsequent query statement. If no column name is provided, the column name is obtained from the output column name of the query.
Value range: a string. It must comply with the [naming convention](#).
- **TABLESPACE tablespace_name**
Tablespace to which the new materialized view belongs. If the tablespace is not specified, the default tablespace is used.
- **AS query**
SELECT or **TABLE** command. This query will be run in a security-constrained operation.

Examples

```
-- Create an ordinary table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int);

-- Create a fast-refresh materialized view.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

-- Write data to the base table.
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Fast refresh the fast-refresh materialized view my_imv.
gaussdb=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

-- Delete a fast-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_imv;

-- Delete the ordinary table my_table.
gaussdb=# DROP TABLE my_table;
```

Helpful Links

[ALTER MATERIALIZED VIEW](#), [CREATE MATERIALIZED VIEW](#), [CREATE TABLE](#), [DROP MATERIALIZED VIEW](#), [REFRESH INCREMENTAL MATERIALIZED VIEW](#), and [REFRESH MATERIALIZED VIEW](#)

7.14.60 CREATE INDEX

Description

Defines a new index.

Indexes are primarily used to enhance database performance (though inappropriate use can result in database performance deterioration). You are advised to create indexes on:

- Columns that are often queried
- Join conditions. For a query on joined columns, you are advised to create a composite index on the columns. For example, for **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**, you can create a composite index on columns **a** and **b** in table **t1**.
- Columns having filter criteria (especially scope criteria) of a **where** clause
- Columns that appear after **order by**, **group by**, and **distinct**

Partitioned tables do not support partial index creation (when indexes contain the GLOBAL or LOCAL keyword or the created index is a GLOBAL index).

Precautions

- If the base table is a hash-distributed table, to create a primary key or unique index that does not contain the distribution key of the base table, use a GSI (using CREATE GLOBAL INDEX); to create a primary key or unique index that contains the distribution key of the base table, use an ordinary index (using CREATE INDEX). In single-DN deployment mode, both the GSI and ordinary index can be successfully created. If the base table is a non-hash-distributed table, you can only create the primary key or unique index as an ordinary index (using CREATE INDEX). That is, the index key must contain the distribution key of the base table.
- Indexes consume storage and computing resources. Creating too many indexes has negative impact on database performance (especially the performance of data import. Therefore, you are advised to import the data before creating indexes). Therefore, create indexes only when they are necessary.
- All functions and operators used in an index definition must be immutable, that is, their results must depend only on their parameters and never on any outside influence (such as the contents of another table or the current time). This restriction ensures that the behavior of the index is well-defined. To use a user-defined function in an index or WHERE clause, mark it as an immutable function.
- A user granted with the CREATE ANY INDEX permission can create indexes in both the public and user schemas.
- If a user-defined function is called in the expression index, the expression index function is executed based on the permission of the function creator.
- Data of the XML type cannot be used as ordinary indexes, unique indexes, global indexes, local indexes, or partial indexes.
- Only B-tree and UB-tree indexes can be created online. Only ordinary indexes of non-partitioned tables, as well as global and local indexes of partitioned tables can be created. Online index column addition, deletion, and modification, PCR UB-tree indexes, level-2 partitions, and GSIs are not supported. Online concurrent index creation supports only Astore ordinary indexes, global indexes, and local indexes. Ustore indexes are not supported.
- Creating an index using CREATE INDEX may change the table access mode. As a result, the query execution plan changes.
- When creating a composite index, you need to create it based on query conditions and the leftmost match principle of the composite index.

 NOTE

- Leftmost match principle of a composite index: If a query condition contains one or more columns of a composite index, the leftmost consecutive columns of the composite index must match the query condition.
- When the query is **WHERE a = ?, b = ?, c = ?, d = ?** or **WHERE a = ?, b = ?, c = ?, d = ?** or **WHERE b = ?, c = ?, d = ?** or **WHERE c = ?, d = ?**, the **idx_test_abcd** index might still be used after cost calculation. In such cases, the index scan will involve all pages of the index, resulting in unsatisfactory SQL performance. In similar cases, you are advised to create a composite index suitable for the query condition based on the leftmost match principle.

```
-- Create the test table.
gaussdb=# CREATE TABLE test(a int, b int, c int, d int, e int, f text);
Create a composite index.
gaussdb=# CREATE INDEX idx_test_abcd ON test(a,b,c,d);
```

Syntax

- Create an index on a table.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [schema_name.] index_name ] ON table_name
[ USING method ]
( ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
{ FIRST | LAST } ] }, ... )
[ INCLUDE ( column_name [, ...] ) ]
[ WITH ( {storage_parameter = value} [, ...] ) ]
[ TABLESPACE tablespace_name ]
[ WHERE predicate ];
```
- Create an index on a partitioned table.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [schema_name.] index_name ] ON table_name
[ USING method ]
( ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
LAST ] }, ... )
[ LOCAL [ ( { PARTITION index_partition_name [ TABLESPACE index_partition_tablespace ] }
[, ...] ) ] | GLOBAL ]
[ INCLUDE ( column_name [, ...] ) ]
[ WITH ( { storage_parameter = value } [, ...] ) ]
[ TABLESPACE tablespace_name ];
```

Parameters

- **UNIQUE**
Creates a unique index. In this way, the system checks whether new values are unique in the index column. Attempts to insert or update data which would result in duplicate entries will generate an error.
Currently, only B-tree indexes and UB-tree indexes in row-store tables support unique indexes.
- **CONCURRENTLY**
Creates an index (with ShareUpdateExclusiveLock) in a mode that does not block DML statements. When an index is created, other statements cannot access the table on which the index depends. If this keyword is specified, DML is not blocked during the creation.
 - This option can only specify a name of one index.
 - The CREATE INDEX statement can be run within a transaction, but CREATE INDEX CONCURRENTLY cannot.
 - For temporary tables, you can use CONCURRENTLY to create indexes. However, indexes are created in blocking mode because no other sessions

concurrently access the temporary tables and the blocking mode is more cost-effective.

 NOTE

- This keyword is specified when an index is created. The entire table needs to be scanned twice and built. When the table is scanned for the first time, an index is created and the read and write operations are not blocked. During the second scan, changes that have occurred since the first scan are merged and updated.
- The table needs to be scanned and built twice, and all existing transactions that may modify the table must be completed. This means that the creation of the index takes a longer time than normal. In addition, the CPU and I/O consumption also affects other services.
- If an index build fails, it leaves an "unusable" index. This index is ignored by queries, but it still consumes the update overhead. In this case, you are advised to run the **DROP INDEX IF EXISTS** statement to delete the index and run the **CONCURRENTLY** statement to create the index again. Note that when indexes are rebuilt by using **CLUSTER**, **TRUNCATE**, **VACUUM FULL**, or **REINDEX TABLE**, the residual "unusable" indexes are skipped. If **ALTER TABLE** involves table and index rebuild, residual "unusable" indexes will be automatically cleared.
- After the second scan, index creation must wait for any transaction that holds a snapshot earlier than the snapshot taken by the second scan to terminate. In addition, the ShareUpdateExclusiveLock (level 4) added during index creation conflicts with a lock whose level is greater than or equal to 4. Therefore, when such an index is created, the system is prone to hang or deadlock. Example:
 - If two sessions create an index by using **CONCURRENTLY** for the same table, a deadlock occurs.
 - If a session creates an index by using **CONCURRENTLY** for a table and another session drops a table, a deadlock occurs.
 - There are three sessions. Session 1 locks table **a** and does not commit it. Session 2 creates an index by using **CONCURRENTLY** for table **b**. Session 3 writes data to table **a**. Before the transaction of session 1 is committed, session 2 is blocked.
 - When an index is created by using **CONCURRENTLY** for a table concurrently with the **TRUNCATE** operation on the same table, a deadlock occurs.
 - The transaction isolation level is set to repeatable read (read committed by default). Two sessions are started. Session 1 writes data to table **a** and does not commit it. Session 2 creates an index by using **CONCURRENTLY** for table **b**. Before the transaction of session 1 is committed, session 2 is blocked.
- When an index is being created or fails to be created, you need to check the index progress or status. You can query the **gs_get_index_status('schema_name', 'index_name')** function to check the index status on all nodes. The input parameters **schema_name** and **index_name** are used to specify the index schema name and index name, respectively. The return values are **node_name**, **indisready**, and **indisvalid**, indicating the node name, whether the index can be inserted on the node, and whether the index is available on the node. The index is available only when **indisready** and **indisvalid** on all nodes are set to **true**; otherwise, wait until the index creation is complete. If the index fails to be created, delete the index and create it again.
- When the I/O and CPU resources are not limited, the service performance deterioration caused by online index creation can be controlled within 10%. However, in special scenarios, the service performance deterioration may exceed 10%. This is because online index creation is a long transaction that consumes a large number of I/O and CPU resources. It consumes more resources than offline index creation. The longer the online index creation transaction lasts, the greater the impact on service performance. The time for creating indexes online is positively correlated with the data volume of base tables and the data volume generated by concurrent DML statements. When the I/O and CPU resources are not limited, the time for creating indexes online is about two to six times that for creating indexes offline. However, when the number of concurrent transactions is large (> 10000 TPS) or resource contention occurs, the time may be even longer. In

Astore mode, you can create indexes in parallel to shorten the index creation time. The performance of online parallel index creation increases to a certain value and becomes stable as the number of parallel worker threads increases. Compared with the performance of creating indexes in serial mode, the performance of creating indexes in parallel online is improved by about 30%. You are advised to create indexes online during off-peak hours to avoid great impact on services. Although online index creation provides the capability of uninterrupted services to some extent, it still needs to be implemented with caution.

- **schema_name**
Specifies the schema name.
Value range: an existing schema name
- **index_name**
Specifies the name of the index to be created. No schema name can be included here; the index is always created in the same schema as its parent table.
Value range: a string. It must comply with the [naming convention](#).
- **table_name**
Specifies the name of the table to be indexed (optionally schema-qualified).
Value range: an existing table name
- **USING method**
Specifies the name of the index method to be used.
Value range:
 - **btree**: B-tree indexes store key values of data in a B+ tree structure. This structure helps users to quickly search for indexes. B-tree supports comparison query and query range. When an index is created in a Ustore table, the index is automatically changed to UB-tree.
 - **ubtree**: Multi-version B-tree index used only for Ustore tables. The index page contains transaction information and can be recycled. By default, the INSERTPT function is enabled for UB-tree indexes.Row-store tables support the following index types: **ubtree** (default). Row-store tables (Ustore) support the following index type: UB-tree.
- **column_name**
Specifies the name of the column on which an index is to be created.
Multiple columns can be specified if the index method supports multi-column indexes. A global index supports a maximum of 31 columns, and other indexes support a maximum of 32 columns.
- **expression**
Specifies an expression based on one or more columns of the table. The expression usually must be written with surrounding parentheses, as shown in the syntax. However, the parentheses can be omitted if the expression has the form of a function call.
Expression can be used to obtain fast access to data based on some transformation of the basic data. For example, an index computed on **upper(col)** would allow the clause **WHERE upper(col) = 'JIM'** to use an index.
If an expression contains **IS NULL**, the index for this expression is invalid. In this case, you are advised to create a partial index.

- **COLLATE collation**

Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **select * from pg_collation** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.
- **opclass**

Specifies the name of an operator class. An operator class can be specified for each column of an index. The operator class identifies the operators to be used by the index for that column. For example, a B-tree index on the type int4 would use the **int4_ops** class; this operator class includes comparison functions for values of type int4. In practice, the default operator class for the column's data type is sufficient. The operator class applies to data with multiple sorts. For example, if you want to sort complex numbers by absolute value or real number, you can define two operator classes and select a proper class when creating an index. In addition, if the value of **COLLATE** of an index of the string type (such as varchar, varchar2, and text) is not **C** or **POSIX**, but you want the index to support prefix matching, you need to specify the **varchar_pattern_ops** option.
- **ASC**

Specifies an ascending (default) sort order.
- **DESC**

Specifies a descending sort order.
- **NULLS FIRST**

Specifies that null values appear before non-null values in the sort ordering. This is the default when **DESC** is specified.
- **NULLS LAST**

Specifies that null values appear after non-null values in the sort ordering. This is the default when **DESC** is not specified.
- **WITH ({storage_parameter = value} [, ...])**

Specifies the storage parameter used for an index.
Value range:
For indexes other than Psort, you can also set it to **FILLFACTOR**. Only non-partitioned B-tree indexes support the **DEDUPLICATION** parameter.

 - **FILLFACTOR**

The fill factor of an index is a percentage from 10 to 100. In the scenario where a large number of concurrent insertions are performed and the key value range is dense, select a smaller fill factor when the contention of the same index page is high during the insertion.
Value range: 10–100
 - **ACTIVE_PAGES**

Specifies the number of index pages, which may be less than the actual number of physical file pages and can be used for optimization. Currently, this parameter is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM and ANALYZE (including AUTOVACUUM). You are advised not to manually set this parameter because it is invalid in distributed mode.

- **DEDUPLICATION**

Specifies whether to deduplicate and compress tuples with duplicate key values. This is an index parameter. When there are a large number of indexes with duplicate key values, enabling this parameter can effectively reduce the space occupied by indexes. This parameter does not take effect for primary key indexes and unique indexes. If non-unique indexes are used and the index key value duplication rate is low or the index key values are unique, enabling this parameter will slightly deteriorate the index insertion performance. Currently, local and global indexes of partitioned tables are not supported.

Value range: Boolean value. The default value is the value of the **enable_default_index_deduplication** GUC parameter (the default value is **off**).

- **TABLESPACE tablespace_name**

Specifies the tablespace for an index. If no tablespace is specified, the default tablespace is used.

Value range: an existing tablespace name.

- **WHERE predicate**

Creates a partial index. A partial index is an index that contains entries for only a portion of a table, usually a portion that is more useful for indexing than the rest of the table. For example, if you have a table that contains both billed and unbilled orders where the unbilled orders take up a small fraction of the total table and yet that is an often used portion, you can improve performance by creating an index on just that portion. In addition, **WHERE** with **UNIQUE** can be used to enforce uniqueness over a subset for a table.

Value range: The predicate expression can only refer to columns of the underlying table, but it can use all columns, not just the ones being indexed. Currently, subqueries and aggregate expressions are forbidden in **WHERE**. You are advised not to use numeric types such as int for predicate, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

For a partitioned table index, if the created index contains the GLOBAL or LOCAL keyword or the created index is a GLOBAL index, the WHERE clause cannot be used to create an index.

- **PARTITION index_partition_name**

Specifies the name of an index partition.

Value range: a string. It must comply with the [naming convention](#).

- **TABLESPACE index_partition_tablespace**

Specifies the tablespace of an index partition.

Value range: If this parameter is not specified, the value of **index_tablespace** is used.

Examples

```
-- Create the tpcds.ship_mode_t1 table.
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# CREATE TABLE tpcds.ship_mode_t1
(
  SM_SHIP_MODE_SK      INTEGER      NOT NULL,
```

```
SM_SHIP_MODE_ID      CHAR(16)      NOT NULL,
SM_TYPE              CHAR(30)
SM_CODE              CHAR(10)
SM_CARRIER          CHAR(20)
SM_CONTRACT          CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);

-- Create a common unique index on the SM_SHIP_MODE_SK column in the tpcds.ship_mode_t1 table.
gaussdb=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

-- Create a B-tree index on the SM_SHIP_MODE_SK column in the tpcds.ship_mode_t1 table.
gaussdb=# CREATE INDEX ds_ship_mode_t1_index4 ON tpcds.ship_mode_t1 USING
btree(SM_SHIP_MODE_SK);

-- Create an expression index on the SM_CODE column in the tpcds.ship_mode_t1 table:
gaussdb=# CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE,1,4));

-- Create a partial index on the SM_SHIP_MODE_SK column where SM_SHIP_MODE_SK is greater than 10
in the tpcds.ship_mode_t1 table.
gaussdb=# CREATE UNIQUE INDEX ds_ship_mode_t1_index3 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK)
WHERE SM_SHIP_MODE_SK>10;

-- Create an index on the SM_SHIP_MODE_SK column of table tpcds.ship_mode_t1 in a mode that does
not block DML.
gaussdb=# CREATE INDEX CONCURRENTLY ds_ship_mode_t1_index4 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

-- Rename an existing index.
gaussdb=# ALTER INDEX tpcds.ds_ship_mode_t1_index1 RENAME TO ds_ship_mode_t1_index5;

-- Set the index as unusable.
gaussdb=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 UNUSABLE;

-- Rebuild an index.
gaussdb=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 REBUILD;

-- Delete an existing index.
gaussdb=# DROP INDEX tpcds.ds_ship_mode_t1_index2;

-- Delete the table.
gaussdb=# DROP TABLE tpcds.ship_mode_t1;

-- Create a tablespace.
gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
gaussdb=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
gaussdb=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
gaussdb=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
-- Create the tpcds.customer_address_p1 table.
gaussdb=# CREATE TABLE tpcds.customer_address_p1
(
  CA_ADDRESS_SK      INTEGER      NOT NULL,
  CA_ADDRESS_ID      CHAR(16)      NOT NULL,
  CA_STREET_NUMBER   CHAR(10)
  CA_STREET_NAME     VARCHAR(60)
  CA_STREET_TYPE     CHAR(15)
  CA_SUITE_NUMBER    CHAR(10)
  CA_CITY            VARCHAR(60)
  CA_COUNTY          VARCHAR(30)
  CA_STATE           CHAR(2)
  CA_ZIP             CHAR(10)
  CA_COUNTRY         VARCHAR(20)
  CA_GMT_OFFSET      DECIMAL(5,2)
  CA_LOCATION_TYPE   CHAR(20)
)
TABLESPACE example1
DISTRIBUTE BY HASH(CA_ADDRESS_SK)
PARTITION BY RANGE(CA_ADDRESS_SK)
(
```

```
PARTITION p1 VALUES LESS THAN (3000),
PARTITION p2 VALUES LESS THAN (5000) TABLESPACE example1,
PARTITION p3 VALUES LESS THAN (MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
-- Create the partitioned table index ds_customer_address_p1_index1 without specifying the index
partition name.
gaussdb=# CREATE INDEX ds_customer_address_p1_index1 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL;
-- Create the partitioned table index ds_customer_address_p1_index2 with the name of the index partition
specified.
gaussdb=# CREATE INDEX ds_customer_address_p1_index2 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL
(
PARTITION CA_ADDRESS_SK_index1,
PARTITION CA_ADDRESS_SK_index2 TABLESPACE example3,
PARTITION CA_ADDRESS_SK_index3 TABLESPACE example4
)
TABLESPACE example2;

-- Create the partitioned table index ds_customer_address_p1_index3 online without specifying the index
partition name.
gaussdb=# CREATE INDEX CONCURRENTLY ds_customer_address_p1_index3 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL;

-- Create the global partitioned index ds_customer_address_p1_index4 online.
gaussdb=# CREATE INDEX CONCURRENTLY ds_customer_address_p1_index4 ON
tpcds.customer_address_p1(CA_ADDRESS_ID) GLOBAL;

-- Change the tablespace of the partitioned table index CA_ADDRESS_SK_index2 to example1.
gaussdb=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION CA_ADDRESS_SK_index2
TABLESPACE example1;

-- Change the tablespace of the partitioned table index CA_ADDRESS_SK_index3 to example2.
gaussdb=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION CA_ADDRESS_SK_index3
TABLESPACE example2;

-- Rename a partitioned table index.
gaussdb=# ALTER INDEX tpcds.ds_customer_address_p1_index2 RENAME PARTITION
CA_ADDRESS_SK_index1 TO CA_ADDRESS_SK_index4;

-- Delete the created indexes and the partitioned table.
gaussdb=# DROP INDEX tpcds.ds_customer_address_p1_index1;
gaussdb=# DROP INDEX tpcds.ds_customer_address_p1_index2;
gaussdb=# DROP TABLE tpcds.customer_address_p1;
-- Delete the tablespace.
gaussdb=# DROP TABLESPACE example1;
gaussdb=# DROP TABLESPACE example2;
gaussdb=# DROP TABLESPACE example3;
gaussdb=# DROP TABLESPACE example4;
```

Helpful Links

[ALTER INDEX](#) and [DROP INDEX](#)

Suggestions

- create index

You are advised to create indexes on:

- Columns that are often queried
- Join conditions. For a query on joined columns, you are advised to create a composite index on the columns. For example, for **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**, you can create a composite index on columns **a** and **b** in table **t1**.

- Columns having filter criteria (especially scope criteria) of a **where** clause
- Columns that appear after **order by**, **group by**, and **distinct**

Constraints:

- An index of an ordinary table supports a maximum of 32 columns. A GLOBAL index of a partitioned table supports a maximum of 31 columns.
- The size of a single index cannot exceed the size of the index page (8 KB). The size of a B-tree or UB-tree index cannot exceed one-third of the page size.
- Partial indexes cannot be created in a partitioned table.

7.14.61 CREATE LANGUAGE

This version does not support this syntax.

7.14.62 CREATE MASKING POLICY

Description

Creates a masking policy.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

The masking policy takes effect only after the security policy is enabled, that is, **enable_security_policy** is set to **on**.

NOTICE

When you use database links to perform operations on remote objects, the client initiates a database link request. The actual sender is the server, and the attributes such as the IP address of the sender are the values of the server. For details, see [DATABASE LINK](#).

Syntax

```
CREATE MASKING POLICY policy_name masking_clause[, ...] [ policy_filter_clause ] [ENABLE | DISABLE];
```

- **masking_clause**
masking_function ON LABEL(label_name[, ...])

- **masking_function**

maskall is not a preset function. It is hard-coded and cannot be displayed by running **\df**.

The preset masking methods are as follows:

```
{ maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |  
shufflemasking | alldigitmasking | regexpmasking }
```

- **policy_filter_clause**:
FILTER ON { FILTER_TYPE (filter_value [, ...]) } [, ...]
- **FILTER_TYPE**:

IP | APP | ROLES

Parameters

- **policy_name**
Specifies the audit policy name, which must be unique.
Value range: a string. It must comply with the [naming convention](#).
- **label_name**
Specifies the resource label name.
- **masking_clause**
Specifies the masking function to be used to anonymize database resources labeled by **label_name**. **schema.function** can be used to specify the masking function.
- **policy_filter**
Specifies the users for which the masking policy takes effect. If this parameter is left empty, the masking policy takes effect for all users.
- **FILTER_TYPE**
Specifies the types of information to be filtered by the policy, including **IP**, **APP**, and **ROLES**.
- **filter_value**
Indicates the detailed information to be filtered, such as the IP address, app name, and username.
- **ENABLE|DISABLE**
Enables or disables the masking policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

```
-- Create users dev_mask and bob_mask.
gaussdb=# CREATE USER dev_mask PASSWORD '*****';
gaussdb=# CREATE USER bob_mask PASSWORD '*****';

-- Create table tb_for_masking.
gaussdb=# CREATE TABLE tb_for_masking(idx int, col1 text, col2 text, col3 text, col4 text, col5 text, col6
text, col7 text,col8 text);
gaussdb=# INSERT INTO tb_for_masking VALUES(1, '9876543210', 'usr321usr', 'abc@huawei.com',
'abc@huawei.com', '1234-4567-7890-0123', 'abcdef 123456 ui 323 jsfd321 j3k2l3', '4880-9898-4545-2525',
'this is a llt case');
gaussdb=# INSERT INTO tb_for_masking VALUES(2, '0123456789', 'lltc123llt', 'abc@gmail.com',
'abc@gmail.com', '9876-5432-1012-3456', '1234 abcd ef 56 gh78ijk90lm', '4856-7654-1234-9865','this,is.a!
LLT?case');

-- Create a resource label for sensitive columns.
gaussdb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);
gaussdb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);
gaussdb=# CREATE RESOURCE LABEL mask_lb3 ADD COLUMN(tb_for_masking.col3);
gaussdb=# CREATE RESOURCE LABEL mask_lb4 ADD COLUMN(tb_for_masking.col4);
gaussdb=# CREATE RESOURCE LABEL mask_lb5 ADD COLUMN(tb_for_masking.col5);
gaussdb=# CREATE RESOURCE LABEL mask_lb6 ADD COLUMN(tb_for_masking.col6);
gaussdb=# CREATE RESOURCE LABEL mask_lb7 ADD COLUMN(tb_for_masking.col7);
gaussdb=# CREATE RESOURCE LABEL mask_lb8 ADD COLUMN(tb_for_masking.col8);

-- Create a masking policy.
gaussdb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);
gaussdb=# CREATE MASKING POLICY maskpol2 alldigitsmasking ON LABEL(mask_lb2);
gaussdb=# CREATE MASKING POLICY maskpol3 basicemailmasking ON LABEL(mask_lb3);
```

```

gaussdb=# CREATE MASKING POLICY maskpol4 fullemailmasking ON LABEL(mask_lb4);
gaussdb=# CREATE MASKING POLICY maskpol5 creditcardmasking ON LABEL(mask_lb5);
gaussdb=# CREATE MASKING POLICY maskpol6 shufflemasking ON LABEL(mask_lb6);
gaussdb=# CREATE MASKING POLICY maskpol7 regexprmasking('[\d+]','*',2, 9) ON LABEL(mask_lb7);

-- Create a masking policy that takes effect only for scenarios where users are dev_mask and bob_mask,
the client tool is gsq, and IP addresses are 10.20.30.40 and 127.0.0.0/24.
gaussdb=# CREATE MASKING POLICY maskpol8 randommasking ON LABEL(mask_lb8) FILTER ON
ROLES(dev_mask, bob_mask), APP(gsql), IP('10.20.30.40', '127.0.0.0/24');
-- Check whether the data masking policy takes effect.
gaussdb=# SELECT * FROM tb_for_masking;
 idx | col1 | col2 | col3 | col4 | col5 | col6 |
-----+-----+-----+-----+-----+-----+-----
 1 | xxxxxxxxxx | usr000usr | xxx@huawei.com | xxx@xxxxxx.com | xxxx-xxxx-xxxx-0123 | s
2iju1bcjk243df333d61l 22 53ef3a | 48**_***_*545-2525 | this is a llt case
 2 | xxxxxxxxxx | lltc000llt | xxx@gmail.com | xxx@xxxxxx.com | xxxx-xxxx-xxxx-3456 | j 71fem0l286dbia543
g k9 ch | 48**_***_*234-9865 | this,is,a!LLT?case
(2 rows)
-- Use the gsql tool, set the IP address to 10.20.30.40, and view tb_for_masking as user dev_mask.
gaussdb=# GRANT ALL PRIVILEGES TO dev_mask;
gaussdb=# GRANT ALL PRIVILEGES TO bob_mask;
gaussdb=# SET role dev_mask PASSWORD '*****';
-- Use maskpol8 for data masking. The result is random and different each time.
gaussdb=# SELECT col8 FROM tb_for_masking;
 col8
-----
9f1425b3835cc30d99
9585b4ea8ea8ddcc5b
(2 rows)
gaussdb=# SET role bob_mask PASSWORD '*****';
gaussdb=# SELECT col8 FROM tb_for_masking;
 col8
-----
f29ef3a0769a1f417c
806aa46409482d838f
(2 rows)

-- Delete the masking policy.
gaussdb=# DROP MASKING POLICY maskpol1, maskpol2, maskpol3, maskpol4, maskpol5, maskpol6,
maskpol7, maskpol8;

-- Delete the resource label.
gaussdb=# DROP RESOURCE LABEL mask_lb1, mask_lb2, mask_lb3, mask_lb4, mask_lb5, mask_lb6,
mask_lb7, mask_lb8;

-- Delete the tb_for_masking table.
gaussdb=# DROP TABLE tb_for_masking;

-- Delete the dev_mask and bob_mask users.
gaussdb=# DROP USER dev_mask, bob_mask;

```

Helpful Links

[5.1.13.14.14-ALTER MASKING POLICY](#) and [5.1.13.14.96-DROP MASKING POLICY](#)

7.14.63 CREATE MATERIALIZED VIEW

CREATE MATERIALIZED VIEW creates a complete-refresh materialized view, and you can use **REFRESH MATERIALIZED VIEW** (full refresh) to refresh the data in the materialized view.

CREATE MATERIALIZED VIEW is similar to **CREATE TABLE AS**, but it remembers the query used to initialize the view, so it can refresh data later. A materialized

view has many attributes that are the same as those of a table, but does not support temporary materialized views.

Precautions

- Complete-refresh materialized views cannot be created in temporary tables or global temporary tables.
- Complete-refresh materialized views do not support NodeGroups.
- After a complete-refresh materialized view is created, most DDL operations in the base table are no longer supported.
- IUD operations cannot be performed on complete-refresh materialized views.
- After a complete-refresh materialized view is created, if the base table data changes, you need to run the refresh command to synchronize the materialized view with the base table.
- The Ustore engine does not support the creation and use of materialized views.

Syntax

```
CREATE MATERIALIZED VIEW mv_name
  [ (column_name [, ...] ) ]
  [ WITH ( {storage_parameter = value} [, ...] ) ]
  [ TABLESPACE tablespace_name ]
  AS query;
```

Parameter Description

- **mv_name**
Name (optionally schema-qualified) of the materialized view to be created.
Value range: a string. It must comply with the [naming convention](#).
- **column_name**
Column name in the new materialized view. The materialized view supports specified columns. The number of specified columns must be the same as the number of columns in the result of the subsequent query statement. If no column name is provided, the column name is obtained from the output column name of the query.
Value range: a string. It must comply with the [naming convention](#).
- **WITH (storage_parameter [= value] [, ...])**
Specifies an optional storage parameter for a table or an index. For details, see [CREATE TABLE](#).
- **TABLESPACE tablespace_name**
Tablespace to which the new materialized view belongs. If the tablespace is not specified, the default tablespace is used.
- **AS query**
SELECT, **TABLE**, or **VALUES** command This query will be run in a security-constrained operation.

Examples

```
-- Create an ordinary table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int);
```

```
-- Create a complete-refresh materialized view.
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

-- Write data to the base table.
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Completely refresh the complete-refresh materialized view my_mv.
gaussdb=# REFRESH MATERIALIZED VIEW my_mv;

-- Delete a complete-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_mv;

-- Delete the ordinary table my_table.
gaussdb=# DROP TABLE my_table;
```

Helpful Links

[ALTER MATERIALIZED VIEW](#), [CREATE INCREMENTAL MATERIALIZED VIEW](#), [CREATE TABLE](#), [DROP MATERIALIZED VIEW](#), [REFRESH INCREMENTAL MATERIALIZED VIEW](#), and [REFRESH MATERIALIZED VIEW](#)

7.14.64 CREATE MODEL

This syntax is not supported in distributed scenarios.

7.14.65 CREATE NODE

Description

Creates a cluster node.

Precautions

CREATE NODE is an interface of the cluster management tool. You are not advised to use this interface, because doing so affects the cluster. Only the administrator has the permission to use this interface.

Syntax

```
CREATE NODE nodename WITH
(
  [ TYPE = nodetype,]
  [ HOST = hostname,]
  [ PORT = portnum,]
  [ HOST1 = 'hostname',]
  [ PORT1 = portnum,]
  [ HOSTPRIMARY [ = boolean ],]
  [ PRIMARY [ = boolean ],]
  [ PREFERRED [ = boolean ],]
  [ SCTP_PORT = portnum,]
  [ CONTROL_PORT = portnum,]
  [ SCTP_PORT1 = portnum,]
  [ CONTROL_PORT1 = portnum ]
);
```

Parameters

- **nodename**
Specifies the node name.

Value range: a string. It must comply with the [naming convention](#).

- **TYPE = nodetype**

Specifies the type of a node.

Value range:

- 'coordinator'
- 'datanode'

- **HOST = hostname**

Specifies the primary server name or IP address of a node.

- **PORT = portnum**

Specifies the primary server port to which a node is bound.

- **HOST1 = hostname**

Specifies the name or IP address of the standby server of a node.

- **PORT1 = portnum**

Specifies the port number of the standby server to which a node is bound.

- **HOSTPRIMARY**

- **PRIMARY = boolean**

Specifies whether the node is a primary node or not. A primary node allows read/write operations. A non-primary node allows only read operations.

Value range:

- **true**
- **false** (default value)

- **PREFERRED = boolean**

Specifies whether the node is a preferred node for read operations.

Value range:

- **true**
- **false** (default value)

- **SCTP_PORT = portnum**

Specifies the port used by the TCP proxy communications library of the primary node to listen on the data transmission channel. TCP is used to listen on connections.

- **CONTROL_PORT = portnum**

Specifies the port used by the TCP proxy communications library of the primary server to listen on the control channel. It may be a TCP port.

- **SCTP_PORT1 = portnum**

Specifies the port used by the TCP proxy communications library of the standby node to listen on the data transmission channel. TCP is used to listen on connections.

- **CONTROL_PORT 1= portnum**

Specifies the port used by the TCP proxy communications library of the standby server to listen on the control channel. It is a TCP port.

Helpful Links

[ALTER NODE](#) and [DROP NODE](#)

7.14.66 CREATE NODE GROUP

Description

Creates a cluster node group.

Precautions

- CREATE NODE GROUP is an interface of the cluster management tool.
- Only a system administrator has the permission.

Syntax

```
CREATE NODE GROUP groupname  
  WITH ( nodename [, ... ] ) [bucketcnt bucket_cnt]  
  [ BUCKETS [ ( bucketnumber [, ... ] ) ] ] [VCGROUP] [DISTRIBUTE FROM src_group_name] [groupparent  
parent_group_name];
```

Parameters

- **groupname**
Specifies the name of a node group.
Value range: a string. It must comply with the [naming convention](#). A value can contain a maximum of 63 characters.
 **NOTE**
A node group name supports all ASCII characters, but you are advised to name a node group according to the naming convention.
- **nodename**
Specifies the node name.
Value range: a string. It must comply with the [naming convention](#). A value can contain a maximum of 63 characters.
If this parameter is not specified, the value of **bucketcnt** must be specified, indicating that child node groups belonging to the installation node group will be created.
- **bucketcnt bucket_cnt**
bucket_cnt indicates the number of buckets.
The value range is [32,16384) and the value must be a power of 2.
If this parameter is not specified, the value of **WITH** must be specified.
- **BUCKETS [(bucketnumber [, ...])]**
Designed for internal use of the cluster management tool. You are advised not to use it directly. Otherwise, the cluster may be affected.
- **groupparent parent_group_name**
parent_group_name indicates the name of the parent node group to which the current child node group belongs.

Helpful Links

[DROP NODE GROUP](#)

7.14.67 CREATE PROCEDURE

Description

Creates a stored procedure.

Precautions

- If the parameters or return values of a stored procedure have precision, the precision is not checked.
- When creating a stored procedure, you are advised to explicitly specify the schemas of all operations on table objects in the stored procedure definition. Otherwise, the stored procedure may fail to be executed.
- **current_schema** and **search_path** specified by **SET** during stored procedure creation are invalid. **search_path** and **current_schema** before and after function execution should be the same.
- When the function is called by SELECT or CALL, an argument must be provided in the output parameter. The argument does not take effect.
- Do not create an overloaded stored procedure with different parameter names but same stored procedure name and parameter list type.
- Do not create a stored procedure that has the same name and parameter list as the function.
- Do not overload stored procedures with different default values.
- Only the IN, OUT, and INOUT parameters of the stored procedure cannot be reloaded after the GUC parameter **behavior_compat_options** is set to '**proc_outparam_override**'. They can be overloaded after the parameter is disabled.
- When an overloaded stored procedure is called, the variable type must be specified.
- If an undeclared variable is used in a stored procedure, an error is reported when the stored procedure is called.
- When you create a procedure, you cannot insert aggregate functions or other functions out of the average function.
- If a function is defined as **IMMUTABLE** or **SHIPPABLE**, avoid **INSERT**, **UPDATE**, **DELETE**, **MERGE**, and **DDL** operations in the function because the CN needs to determine the execution node for these operations. Otherwise, an error may occur.
- The stored procedure does not support operations that will return a set.
- When stored procedures without parameters are called in another stored procedure, you can omit brackets and call stored procedures using their names directly.
- When other functions with output parameters are called in a stored procedure and an assignment expression, set the GUC parameter **behavior_compat_options** to '**proc_outparam_override**', define variables of the same type as the output parameters in advance, and use the variables as

output parameters to call other functions with output parameters for the output parameters to take effect. Otherwise, the output parameters of the called function will be ignored.

- The stored procedure supports viewing, exporting, and importing parameter comments.
- The stored procedure supports viewing, exporting, and importing parameter comments between IS/AS and plsql_body.
- Users granted with the CREATE ANY FUNCTION permission can create or replace stored procedures in the user schemas.
- The default permission on a stored procedure is **SECURITY INVOKER**. To change the default permission to **SECURITY DEFINER**, set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**.
- When a stored procedure is created, it depends on an undefined object. If **behavior_compat_options** is set to **'plpgsql_dependency'**, the creation can be executed, and a warning message is displayed. If **behavior_compat_options** is not set to **'plpgsql_dependency'**, the creation cannot be executed.
- When separation of duties is enabled, the stored procedure of the definer permission can be rebuilt only by the current user.
- If a stored procedure with the definer specified is created in a schema of another user, the stored procedure will be executed by another user, which may cause unauthorized operations. Therefore, exercise caution when performing this operation.
- If the **out** parameter is used as the output parameter in an expression, the expression does not take effect in the following scenarios: (a) The **execute immediate sqlv using func** syntax is used to execute a function. (b) The **select func into** syntax is used to execute a function. (c) DML statements such as INSERT and UPDATE are used to execute a function. (d) The **select where a=func()** statement is used. (e) When a function with the **out** output parameter is used as an input parameter, that is, **fun (func (out b), a)**, the **out b** parameter does not take effect.
- When a stored procedure with the out parameter is called, you can set the GUC parameter **behavior_compat_options** to **'proc_outparam_transfer_length'** to set the parameter length. The specifications and constraints are as follows:
 - a. The following types are supported: CHAR(n), CHARACTER(n), NCHAR(n), VARCHAR(n), VARYING(n), VARCHAR2(n), and NVARCHAR2(n).
 - b. If the out parameter does not take effect (for example, **perform**), the length does not need to be transferred.
 - c. The following types do not support precision transfer: NUMERIC, DECIMAL, NUMBER, FLOAT, DEC, INTEGER, TIME, TIMESTAMP, INTERVAL, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE, TIME WITHOUT TIME ZONE, and TIMESTAMP WITHOUT TIME ZONE.
 - d. The parameter length can be transferred regardless of whether the GUC parameter **set behavior_compat_options** is set to **proc_outparam_override**.

Syntax

```
gaussdb=# CREATE [ OR REPLACE ] PROCEDURE procedure_name  
[ ( ( [ argname ] [ argmode ] argtype [ { DEFAULT | := | = } expression ] ) [... ] ) ]
```

```
[
  { IMMUTABLE | STABLE | VOLATILE }
  | { SHIPPABLE | NOT SHIPPABLE }
  | [ NOT ] LEAKPROOF
  | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
  | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER | AUTHID
CURRENT_USER}
  | COST execution_cost
  | SET configuration_parameter { [ TO | = ] value | FROM CURRENT }
][ ... ]
{ IS | AS }
plsql_body
/
```

Parameters

- **OR REPLACE**
Replaces the original definition when two stored procedures are with the same name.
- **procedure_name**
Specifies the name of the stored procedure that is created (optionally with schema names).
Value range: a string. It must comply with the [naming convention](#).
- **argmode**
Specifies the mode of an argument.

NOTICE

VARIADIC specifies parameters of the array type.

Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**. The default value is **IN**. Only the parameter of the **OUT** mode can be followed by **VARIADIC**. The parameters of **OUT** and **INOUT** cannot be used in procedure definition of **RETURNS TABLE**.

- **argname**
Specifies the argument name.
Value range: a string. It must comply with the [naming convention](#).
- **argtype**
Specifies the type of an argument. You can use **%ROWTYPE** to indirectly reference the type of a table.
Value range: a valid data type
- **expression**
Specifies the default expression of a parameter.

 NOTE

- If **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, the default expression is not supported when the parameter is in INOUT mode.
 - It is recommended that you define all default parameters after all non-default parameters.
 - If a function with default parameters is called, input parameters are added to the function from left to right. If inputs of non-default parameters are missing, an error is reported.
 - If **proc_uncheck_default_param** is enabled and a function with default parameters is called, input parameters are added to the function from left to right. If an input of a non-default parameter is missing, the previous default value is used to fill this parameter.
 - When **a_format_version** is set to **10c**, **a_format_dev_version** is set to **s1**, **proc_outparam_override** is disabled, and the function parameters include the output parameter **out** and **default**, the default value cannot be used.
- **IMMUTABLE, STABLE,...**
Specifies a constraint. The function of each parameter is similar to that of CREATE FUNCTION. For details, see [CREATE FUNCTION](#).
 - **plsql_body**
Specifies the PL/SQL stored procedure body.

NOTICE

When you perform password-related operations, such as user creation, password change, and encryption/decryption, in a stored procedure, the password will be recorded in the system catalogs and logs in plaintext. To prevent sensitive information leakage, you are advised not to perform password-related operations in a stored procedure.

 NOTE

No specific order is applied to **argname** and **argmode**. The following order is advised: **argname**, **argmode**, and **argtype**.

Examples

```
-- Create a stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE prc_add
(
    param1 IN INTEGER,
    param2 IN OUT INTEGER
)
AS
BEGIN
    param2:= param1 + param2;
    db_output.print_line('result is: '||to_char(param2));
END;
/

-- Call the stored procedure.
gaussdb=# SELECT prc_add(2,3);

-- Create a stored procedure whose parameter type is VARIADIC.
gaussdb=# CREATE OR REPLACE PROCEDURE pro_variadic (var1 VARCHAR2(10) DEFAULT 'hello!',var4
VARIADIC int4[])
AS
```

```
BEGIN
  dbe_output.print_line(var1);
END;
/

-- Execute the stored procedure.
gaussdb=# SELECT pro_variadic(var1=>'hello', VARIADIC var4=> array[1,2,3,4]);

-- Create a stored procedure with the permission of the user who calls it.
gaussdb=# CREATE TABLE tb1(a integer);
gaussdb=# CREATE PROCEDURE insert_data(v integer)
SECURITY INVOKER
AS
BEGIN
  INSERT INTO tb1 VALUES(v);
END;
/

-- Call the stored procedure.
gaussdb=# CALL insert_data(1);

-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE prc_add;
gaussdb=# DROP PROCEDURE pro_variadic;
gaussdb=# DROP PROCEDURE insert_data;
```

Helpful Links

[DROP PROCEDURE](#)

Suggestions

- analyse | analyze
 - Do not run **ANALYZE** in a transaction or anonymous block.
 - Do not run **ANALYZE** in a function or stored procedure.

7.14.68 CREATE RESOURCE LABEL

Description

CREATE RESOURCE LABEL creates a resource label.

Precautions

Only user with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

Syntax

```
CREATE RESOURCE LABEL [IF NOT EXISTS] label_name ADD label_item_list[, ...];
```

- label_item_list
resource_type(resource_path[, ...])
- resource_type
{ TABLE | COLUMN | SCHEMA | VIEW | FUNCTION }

Parameters

- **IF NOT EXISTS**

If a resource label with the same name already exists, no error is generated. Instead, a message is displayed, indicating that the resource label already exists.

- **label_name**
Specifies the resource label name, which must be unique.
Value range: a string. It must comply with the [naming convention](#).
- **resource_type**
Specifies the type of database resources to be labeled.
Value range: **TABLE**, **COLUMN**, **SCHEMA**, **VIEW**, and **FUNCTION**.
- **resource_path**
Specifies the path of database resources.

Examples

```
-- Create table tb_for_label.
gaussdb=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

-- Create schema schema_for_label.
gaussdb=# CREATE SCHEMA schema_for_label;

-- Create view view_for_label.
gaussdb=# CREATE VIEW view_for_label AS SELECT 1;

-- Create function func_for_label.
gaussdb=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$
LANGUAGE SQL;

-- Create a resource label based on a table.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

-- Create a resource label based on columns.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add COLUMN(public.tb_for_label.col1);

-- Create a resource label based on a schema.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

-- Create a resource label based on a view.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

-- Create a resource label based on a function.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);

-- Delete a resource label.
gaussdb=# DROP RESOURCE LABEL func_label, view_label, schema_label, column_label, table_label;

-- Delete the func_for_label function.
gaussdb=# DROP FUNCTION func_for_label;

-- Delete the view_for_label view.
gaussdb=# DROP VIEW view_for_label;

-- Delete the schema_for_label schema.
gaussdb=# DROP SCHEMA schema_for_label;

-- Delete the tb_for_label table.
gaussdb=# DROP TABLE tb_for_label;
```

Helpful Links

[ALTER RESOURCE LABEL](#) and [DROP RESOURCE LABEL](#)

7.14.69 CREATE ROLE

Description

Creates a role.

A role is an entity that owns database objects and permissions. In different environments, a role can be considered a user, a group, or both.

Precautions

- If a role is added to the database, the role does not have the login permission.
- Only the user who has the CREATE ROLE permission or a system administrator is allowed to create roles.

Syntax

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY }- { 'password' [EXPIRED] | DISABLE };
```

The syntax of role information configuration clause option is as follows:

```
{SYSADMIN | NOSYSADMIN}  
| {MONADMIN | NOMONADMIN}  
| {OPRADMIN | NOOPRADMIN}  
| {POLADMIN | NOPOLADMIN}  
| {AUDITADMIN | NOAUDITADMIN}  
| {CREATEDB | NOCREATEDB}  
| {USEFT | NOUSEFT}  
| {CREATEROLE | NOCREATEROLE}  
| {INHERIT | NOINHERIT}  
| {LOGIN | NOLOGIN}  
| {REPLICATION | NOREPLICATION}  
| {VCADMIN | NOVADMIN}  
| {PERSISTENCE | NOPERSISTENCE}  
| CONNECTION LIMIT connlimit  
| VALID BEGIN 'timestamp'  
| VALID UNTIL 'timestamp'  
  
| USER GROUP 'groupuser'  
| PERM SPACE 'spacelimit'  
| TEMP SPACE 'tmpspacelimit'  
| SPILL SPACE 'spillspacelimit'  
| NODE GROUP logic_cluster_name  
| IN ROLE role_name [, ...]  
| IN GROUP role_name [, ...]  
| ROLE role_name [, ...]  
| ADMIN role_name [, ...]  
| USER role_name [, ...]  
| SYSID uid  
| DEFAULT TABLESPACE tablespace_name  
| PROFILE DEFAULT  
| PROFILE profile_name  
| PGUSER
```

Parameters

- **role_name**
Specifies the name of a role.
Value range: a string. It must comply with the [naming convention](#). A value can contain a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the

role name. If a role name contains uppercase letters, the database automatically converts the uppercase letters into lowercase letters. To create a role name that contains uppercase letters, enclose the role name with double quotation marks ("").

- **password**

Specifies the login password.

A new password must:

- Contain at least eight characters. This is the default length.
- Differ from the role name or the role name spelled backward.
- Contain at least three of the following character types: uppercase characters, lowercase characters, digits, and special characters (limited to ~!@#\$%^&*()-_+=\|[{]};:;<.>/?). If the password contains characters other than the preceding characters, an error will be reported during statement execution.
- The password can also be a ciphertext character string that meets the format requirements. This mode is mainly used to import user data. You are advised not to use it directly. If a ciphertext password is used, the user must know the plaintext corresponding to the ciphertext password and ensure that the plaintext password meets the complexity requirements. The database does not verify the complexity of the ciphertext password. Instead, the security of the ciphertext password is ensured by the user.
- Be enclosed by single quotation marks when a role is created.

Value range: a character string that cannot be empty.

- **EXPIRED**

When creating a user, you can select **EXPIRED**. That is, you can create a user whose password is invalid. The user cannot perform simple query or extended query. The statement can be executed only after the password is changed.

- **DISABLE**

By default, you can change your password unless it is disabled. To disable the password of a user, use this parameter. After the password of a user is disabled, the password will be deleted from the system. The user can connect to the database only through external authentication, for example, Kerberos authentication. Only administrators can enable or disable a password. Common users cannot disable the password of an initial user. To enable a password, run **ALTER USER** and specify the password.

- **ENCRYPTED | UNENCRYPTED**

Controls whether the password is stored encrypted in the system catalogs. According to product security requirement, the password must be stored encrypted. Therefore, **UNENCRYPTED** is forbidden in GaussDB. If the password string has already been encrypted in the SHA256 format, it is stored encrypted as it was, regardless of whether **ENCRYPTED** or **UNENCRYPTED** is specified (since the system cannot decrypt the specified encrypted password string). This allows reloading of encrypted passwords during dump/restore.

- **SYSADMIN | NOSYSADMIN**

Specifies whether a new role is a system administrator. Roles with the **SYSADMIN** attribute have the highest permission.

Value range: If not specified, **NOSYSADMIN** is the default.

When separation of duties is enabled, users with the SYSADMIN permission do not have the permission to create users.

- **MONADMIN | NOMONADMIN**
Specifies whether a role is a MONADMIN.
Value range: If not specified, **NOMONADMIN** is the default.
- **OPRADMIN | NOOPRADMIN**
Specifies whether a role is an OPRADMIN.
Value range: If not specified, **NOOPRADMIN** is the default.
- **POLADMIN | NOPOLADMIN**
Specifies whether a role is a POLADMIN.
Value range: If not specified, **NOPOLADMIN** is the default.
- **AUDITADMIN | NOAUDITADMIN**
Specifies whether a role has the audit and management attributes.
If not specified, **NOAUDITADMIN** is the default.
- **CREATEDB | NOCREATEDB**
Specifies a role's permission to create databases.
A new role does not have the permission to create databases.
Value range: If not specified, **NOCREATEDB** is the default.
- **USEFT | NOUSEFT**
This parameter is reserved and not used in this version.
- **CREATEROLE | NOCREATEROLE**
Specifies whether a role will be permitted to create new roles (that is, execute **CREATE ROLE** and **CREATE USER**). A role with the CREATEROLE permission can also modify and delete other roles.
Value range: If not specified, **NOCREATEROLE** is the default.
 - When separation of duties is disabled, users with the CREATEROLE permission can create users with the CREATEROLE, AUDITADMIN, MONADMIN, POLADMIN, or CREATEDB permission and common users.
 - When separation of duties is enabled, users with the CREATEROLE permission can create users with the CREATEROLE, MONADMIN, POLADMIN, or CREATEDB permission and common users.
- **INHERIT | NOINHERIT**
Specifies whether a role "inherits" the permissions of roles in the same group. It is not recommended.
- **LOGIN | NOLOGIN**
Specifies whether a role is allowed to log in to a database. A role having the **LOGIN** attribute can be considered as a user.
Value range: If not specified, **NOLOGIN** is the default.
- **REPLICATION | NOREPLICATION**
Specifies whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. A role having the **REPLICATION** attribute is specific to replication.
If not specified, **NOREPLICATION** is the default.

- **PERSISTENCE | NOPERSISTENCE**
Defines a permanent user. Only the initial user is allowed to create, modify, and delete permanent users with the PERSISTENCE attribute.
- **CONNECTION LIMIT**
Specifies how many concurrent connections the role can make.

NOTICE

- System administrators are not restricted by this parameter.
- **conlimit** is calculated for each CN. The number of connections in a cluster is calculated using the following formula: Number of connections in a cluster = **conlimit** x Number of normal CNs.

Value range: an integer in the range $[-1, 2^{31} - 1]$. The default value is **-1**, which means unlimited.

- **VALID BEGIN**
Sets the timestamp when a role takes effect. If this clause is omitted, the role has no valid start time. **timestamp** indicates the start time. The format is 'YYYY-MM-DD HH:mm:ss'.
- **VALID UNTIL**
Sets a date and time after which the role's password is no longer valid. If this clause is omitted, the role has no valid end time. **timestamp** indicates the end time. The format is 'YYYY-MM-DD HH:mm:ss'.
- **RESOURCE POOL**
Sets the name of resource pool used by the role. The name belongs to the system catalog **pg_resource_pool**.
- **USER GROUP 'groupuser'**
Creates a sub-user.
- **PERM SPACE**
Sets the space available for a user.
- **TEMP SPACE**
Sets the space allocated to the temporary table of a user.
- **SPILL SPACE**
Sets the operator disk flushing space of a user.
- **IN ROLE**
Lists one or more existing roles to which the new role will be immediately added as a new member. It is not recommended.
- **IN GROUP**
Specifies an obsolete spelling of **IN ROLE**. It is not recommended.
- **ROLE**
Lists one or more existing roles which are automatically added as members of the new role.
- **ADMIN**

Similar to **ROLE**. However, **ADMIN** grants permissions of new roles to other roles.

- **USER**
Specifies an obsolete spelling of the **ROLE** clause.
- **SYSID**
The **SYSID** clause is ignored.
- **DEFAULT TABLESPACE**
The **DEFAULT TABLESPACE** clause is ignored.
- **PROFILE**
The **PROFILE** clause is ignored.
- **PGUSER**
In the current version, this attribute is reserved only for forward compatibility.

Examples

```
-- Create a role manager whose password is *****
gaussdb=# CREATE ROLE manager IDENTIFIED BY '*****';

-- Create a role with its validity from January 1, 2015 to January 1, 2026.
gaussdb=# CREATE ROLE miriam WITH LOGIN PASSWORD '*****' VALID BEGIN '2015-01-01' VALID UNTIL
'2026-01-01';

-- Change the password of role manager to *****.
gaussdb=# ALTER ROLE manager IDENTIFIED BY '*****' REPLACE '*****';

-- Change role manager to SYSADMIN.
gaussdb=# ALTER ROLE manager SYSADMIN;

-- Delete role manager.
gaussdb=# DROP ROLE manager;

-- Delete role miriam.
gaussdb=# DROP GROUP miriam;
```

Helpful Links

[SET ROLE](#), [ALTER ROLE](#), [DROP ROLE](#), [GRANT](#), and [REVOKE](#)

7.14.70 CREATE ROW LEVEL SECURITY POLICY

Description

Creates a row-level security policy for a table.

The policy takes effect only after row-level security is enabled (by running **ALTER TABLE... ENABLE ROW LEVEL SECURITY**). Otherwise, this statement does not take effect.

Currently, row-level security affects the read (**SELECT**, **UPDATE**, and **DELETE**) of data tables and does not affect the write (**INSERT** and **MERGE INTO**) of data tables. The table owner or system administrators can create an expression in the **USING** clause. When the client reads the data table, the database server combines the expressions that meet the condition and applies it to the execution plan in the statement rewriting phase of a query. For each tuple in a data table, if the

expression returns **TRUE**, the tuple is visible to the current user; if the expression returns **FALSE** or **NULL**, the tuple is invisible to the current user.

A row-level security policy name is specific to a table. A data table cannot have row-level security policies with the same name. Different data tables can have the same row-level security policy.

Row-level security policies can be applied to specified operations (**SELECT**, **UPDATE**, **DELETE**, and **ALL**). **ALL** indicates that **SELECT**, **UPDATE**, and **DELETE** will be affected. For a new row-level security policy, the default value **ALL** will be used if you do not specify the operations that will be affected.

Row-level security policies can be applied to a specified user (role) or to all users (**PUBLIC**). For a new row-level security policy, the default value **PUBLIC** will be used if you do not specify the user that will be affected.

Precautions

- Row-level security policies can be defined for row-store tables, row-store partitioned tables, replication tables, unlogged tables, and hash tables.
- Row-level security policies cannot be defined for foreign tables and temporary tables.
- Row-level security policies cannot be defined for views.
- A maximum of 100 row-level security policies can be defined for a table.
- System administrators are not affected by row-level security policies and can view all data in a table.
- Tables queried by using SQL statements, views, functions, and stored procedures are affected by row-level security policies.
- The data type of a table column to which a row-level security policy has been added cannot be changed.

Syntax

```
CREATE [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
[ AS { PERMISSIVE | RESTRICTIVE } ]
[ FOR { ALL | SELECT | UPDATE | DELETE } ]
[ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
USING ( using_expression )
```

Parameters

- **policy_name**
Specifies the name of a row-level security policy to be created. The names of row-level security policies for a table must be unique.
- **table_name**
Specifies the name of a table to which a row-level security policy is applied.
- **PERMISSIVE | RESTRICTIVE**
PERMISSIVE enables the permissive row-level security policy. The conditions of the permissive policy are joined through the OR expression. **RESTRICTIVE** enables the restrictive row-level security policy. The conditions of the restrictive policy are joined through the AND expression. The join methods are as follows:

(using_expression_permissive_1 OR using_expression_permissive_2 ...) AND
(using_expression_restrictive_1 AND using_expression_restrictive_2 ...)

The default value is **PERMISSIVE**.

- **command**

Specifies the SQL operations affected by a row-level security policy, including **ALL**, **SELECT**, **UPDATE**, and **DELETE**. If this parameter is not specified, the default value **ALL** will be used, covering **SELECT**, **UPDATE**, and **DELETE**.

If **command** is set to **SELECT**, only tuple data that meets the condition (the return value of **using_expression** is **TRUE**) can be queried. The operations that are affected include **SELECT**, **SELECT FOR UPDATE/SHARE**, **UPDATE ... RETURNING**, and **DELETE ... RETURNING**.

If **command** is set to **UPDATE**, only tuple data that meets the condition (the return value of **using_expression** is **TRUE**) can be updated. The operations that are affected include **UPDATE**, **UPDATE ... RETURNING**, and **SELECT ... FOR UPDATE/SHARE**.

If **command** is set to **DELETE**, only tuple data that meets the condition (the return value of **using_expression** is **TRUE**) can be deleted. The operations that are affected include **DELETE** and **DELETE ... RETURNING**.

The following table describes the relationship between row-level security policies and SQL statements.

Table 7-163 Relationship between row-level security policies and SQL statements.

Command	SELECT/ALL Policy	UPDATE/ALL Policy	DELETE/ALL Policy
SELECT	Existing row	No	No
SELECT FOR UPDATE/SHARE	Existing row	Existing row	No
UPDATE	No	Existing row	No
UPDATE RETURNING	Existing row	Existing row	No
DELETE	No	No	Existing row
DELETE RETURNING	Existing row	No	Existing row

- **role_name**

Specifies database users affected by a row-level security policy.

CURRENT_USER indicates the username in the current execution environment. **SESSION_USER** indicates the session username. If this parameter is not specified, the default value **PUBLIC** is used, indicating that all database users are affected. You can specify multiple affected database users.

NOTICE

System administrators are not affected by row access control.

- **using_expression**

Specifies an expression defined for a row-level security policy (return type: Boolean).

The expression cannot contain aggregate functions or window functions. In the statement rewriting phase of a query, if row-level security for a data table is enabled, the expressions that meet the specified conditions will be added to the plan tree. The expression is calculated for each tuple in the data table. For **SELECT**, **UPDATE**, and **DELETE**, row data is visible to the current user only when the return value of the expression is **TRUE**. If the expression returns **FALSE**, the tuple is invisible to the current user. In this case, the user cannot view the tuple through the **SELECT** statement, update the tuple through the **UPDATE** statement, or delete the tuple through the **DELETE** statement.

Examples

```
-- Create user alice.
gaussdb=# CREATE USER alice PASSWORD '*****!';

-- Create user bob.
gaussdb=# CREATE USER bob PASSWORD '*****!';

-- Create data table all_data.
gaussdb=# CREATE TABLE public.all_data(id int, role varchar(100), data varchar(100));

-- Insert data into the data table.
gaussdb=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
gaussdb=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
gaussdb=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

-- Grant the read permission on the all_data table to users alice and bob.
gaussdb=# GRANT SELECT ON all_data TO alice, bob;

-- Enable the row-level security policy.
gaussdb=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

-- Create a row-level security policy to specify that the current user can view only their own data.
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

-- View information about the all_data table.
gaussdb=# \d+ all_data
          Table "public.all_data"
  Column |          Type          | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
  id     | integer                |           |         |              |
  role   | character varying(100) |           | extended|              |
  data   | character varying(100) |           | extended|              |
Row Level Security Policies:
  POLICY "all_data_rls" FOR ALL
  TO public
  USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Run SELECT.
gaussdb=# SELECT * FROM all_data;
 id | role | data
```

```
-----+-----+-----
 1 | alice | alice data
 2 | bob   | bob data
 3 | peter | peter data
(3 rows)

gaussdb=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
          QUERY PLAN
-----+-----+-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on all_data
(3 rows)

-- Switch to user alice and run SELECT.
gaussdb=# SET ROLE alice PASSWORD '*****';gaussdb=> SELECT * FROM all_data;
 id | role | data
-----+-----+-----
  1 | alice | alice data
(1 row)

gaussdb=> EXPLAIN(COSTS OFF) SELECT * FROM all_data;
          QUERY PLAN
-----+-----+-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on all_data
    Filter: ((role)::name = 'alice'::name)
Notice: This query is influenced by row level security feature
(5 rows)

-- Delete the row-level security policy.
gaussdb=> RESET ROLE;
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

-- Delete the all_data table.
gaussdb=# DROP TABLE public.all_data;

-- Delete users alice and bob.
gaussdb=# DROP USER alice, bob;
```

Helpful Links

[DROP ROW LEVEL SECURITY POLICY](#)

7.14.71 CREATE SCHEMA

Description

Creates a schema.

Named objects are accessed either by "qualifying" their names with the schema name as a prefix, or by setting a search path that includes the desired schema. When creating named objects, you can also use the schema name as a prefix.

Optionally, CREATE SCHEMA can include sub-commands to create objects within the new schema. The sub-commands are treated essentially the same as separate commands issued after creating the schema. If the AUTHORIZATION clause is used, all the created objects are owned by this user.

Precautions

- Only a user with the CREATE permission on the current database can perform this operation.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.

Syntax

- Create a schema based on a specified name.

```
CREATE SCHEMA schema_name  
[ AUTHORIZATION user_name ] [ schema_element [ ... ] ];
```
- Create a schema based on a username.

```
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ];
```

Parameters

- **schema_name**
Specifies the schema name.

NOTICE

- The schema name must be unique in the current database.
- The schema name cannot be the same as the initial username of the current database.
- The schema name cannot start with pg_.
- The schema name cannot start with gs_role_.

Value range: a string. It must comply with the [naming convention](#).

- **AUTHORIZATION user_name**
Specifies the owner of a schema. If **schema_name** is not specified, **user_name** will be used as the schema name. In this case, **user_name** can only be a role name.
Value range: an existing username or role name
- **schema_element**
Specifies an SQL statement defining an object to be created within the schema. Currently, only CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE PARTITION, and GRANT are supported.
Objects created by sub-commands are owned by the user specified by AUTHORIZATION.

NOTE

If objects in the schema on the current search path are with the same name, specify the schemas for different objects. You can run **SHOW SEARCH_PATH** to check the schemas on the current search path.

Examples

```
-- Create the role1 role.  
gaussdb=# CREATE ROLE role1 IDENTIFIED BY '*****';
```

```
-- Create a schema named role1 for the role1 role. The owner of the films and winners tables created by
the clause is role1.
gaussdb=# CREATE SCHEMA AUTHORIZATION role1
CREATE TABLE films (title text, release date, awards text[])
CREATE VIEW winners AS
SELECT title, release FROM films WHERE awards IS NOT NULL;

-- Delete the schema.
gaussdb=# DROP SCHEMA role1 CASCADE;
-- Delete the user.
gaussdb=# DROP USER role1 CASCADE;
```

Helpful Links

[ALTER SCHEMA](#) and [DROP SCHEMA](#)

7.14.72 CREATE SEQUENCE

Function

CREATE SEQUENCE adds a sequence to the current database. The owner of a sequence is the user who creates the sequence.

Precautions

- A sequence is a special table that stores arithmetic columns. Such a table is controlled by DBMS. It has no actual meaning and is usually used to generate unique identifiers for rows or tables.
- If a schema name is given, the sequence is created in the specified schema; otherwise, it is created in the current schema. The sequence name must be different from the names of other sequences, tables, indexes, views in the same schema.
- After the sequence is created, functions **nextval()** and **generate_series(1,N)** insert data to the table. Make sure that the number of times for invoking **nextval** is greater than or equal to N+1. Otherwise, errors will be reported because the number of times for invoking function **generate_series()** is N+1.
- A user granted with the CREATE ANY SEQUENCE permission can create sequences in the public and user schemas.

Syntax

```
CREATE SEQUENCE name [ INCREMENT [ BY ] increment ]
[ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE |
NOMAXVALUE]
[ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
[ OWNED BY { table_name.column_name | NONE } ];
```

Parameter Description

- **name**
Specifies the name of a sequence to be created.
Value range: a sting containing only lowercase letters, uppercase letters, special characters #_\$, and digits
- **increment**

Specifies the step for a sequence. A positive number generates an ascending sequence, and a negative number generates a decreasing sequence.

The default value is **1**.

 **NOTE**

In MySQL compatibility mode, if the step is a floating point number, the value is automatically converted to an integer. In other modes, this parameter cannot be set to a floating point number.

- **MINVALUE minvalue | NO MINVALUE| NOMINVALUE**

Specifies the minimum value of the sequence. If **MINVALUE** is not declared, or **NO MINVALUE** is declared, the default value of the ascending sequence is **1**, and that of the descending sequence is **-2⁶³-1**. **NOMINVALUE** is equivalent to **NO MINVALUE**.

- **MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE**

Specifies the maximum value of the sequence. If **MAXVALUE** is not declared, or **NO MAXVALUE** is declared, the default value of the ascending sequence is **2⁶³-1**, and that of the descending sequence is **-1**. **NOMAXVALUE** is equivalent to **NO MAXVALUE**.

- **start**

Specifies the start value of the sequence. The default value for an ascending sequence is **minvalue** and that for a descending sequence is **maxvalue**.

- **cache**

Specifies the number of sequences stored in the memory for quick access purposes.

Default value **1** indicates that one sequence can be generated each time.

 **NOTE**

- It is not recommended that you define **cache** and **maxvalue** or **minvalue** at the same time. The continuity of sequences cannot be ensured after **cache** is defined because unacknowledged sequences may be generated, causing waste of sequences. If there are requirements on the concurrency performance, see the **session_sequence_cache** parameter.
- **cache** specifies the value that a single CN/DN applies for from the GTM at a time. **session_sequence_cache** specifies the value of the cache that a single session applies for from the CN/DN at a time. The value is automatically discarded after the session ends.
- **CYCLE**
Recycles sequences after the number of sequences reaches **maxvalue** or **minvalue**.
If **NO CYCLE** is specified, any invocation of **nextval** would return an error after the number of sequences reaches **maxvalue** or **minvalue**.
NOCYCLE is equivalent to **NO CYCLE**.
The default value is **NO CYCLE**.
If **CYCLE** is specified, the sequence uniqueness cannot be ensured.
- **OWNED BY-**
Associates a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to. The associated table and sequence must

be owned by the same user and in the same schema. **OWNED BY** only establishes the association between a table column and the sequence. Sequences on the column do not increase automatically when data is inserted.

The default value **OWNED BY NONE** indicates that such association does not exist.

NOTICE

You are not advised to use the sequence created using **OWNED BY** in other tables. If multiple tables need to share a sequence, the sequence must not belong to a specific table.

Examples

Create an ascending sequence named **serial**, which starts from 101.

```
gaussdb=# CREATE SEQUENCE serial
START 101
CACHE 20;
```

Select the next number from the sequence.

```
gaussdb=# SELECT nextval('serial');
nextval
-----
      101
```

Select the next number from the sequence.

```
gaussdb=# SELECT nextval('serial');
nextval
-----
      102
```

Create a sequence associated with the table.

```
gaussdb=# CREATE TABLE customer_address
(
  ca_address_sk      integer      not null,
  ca_address_id     char(16)     not null,
  ca_street_number  char(10)
  ,
  ca_street_name    varchar(60)
  ,
  ca_street_type    char(15)
  ,
  ca_suite_number   char(10)
  ,
  ca_city           varchar(60)
  ,
  ca_county         varchar(30)
  ,
  ca_state          char(2)
  ,
  ca_zip            char(10)
  ,
  ca_country        varchar(20)
  ,
  ca_gmt_offset     decimal(5,2)
  ,
  ca_location_type  char(20)
);

gaussdb=# CREATE SEQUENCE serial1
START 101
CACHE 20
OWNED BY customer_address.ca_address_sk;
-- Delete the sequence.
gaussdb=# DROP TABLE customer_address;
gaussdb=# DROP SEQUENCE serial cascade;
gaussdb=# DROP SEQUENCE serial1 cascade;
```

Helpful Links

[DROP SEQUENCE](#) and [ALTER SEQUENCE](#)

7.14.73 CREATE SERVER

Description

Creates a foreign server.

A foreign server stores other homogeneous cluster information.

Precautions

Only a system administrator and users with permission to use a specified **FOREIGN DATA WRAPPER** can create a foreign server. The authorization syntax is as follows:

```
GRANT USAGE ON FOREIGN DATA WRAPPER fdw_name TO username
```

fdw_name is the name of the FOREIGN DATA WRAPPER, and **username** is the name of the user creating a foreign server.

When multi-layer quotation marks are used for sensitive columns (such as **password**) in **OPTIONS**, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

```
CREATE SERVER server_name  
  FOREIGN DATA WRAPPER fdw_name  
  [ OPTIONS ( { option_name ' value ' } [, ...] ) ] ;
```

Parameters

- **server_name**
Specifies the server name.
Value range: a string containing no more than 63 characters
- **FOREIGN DATA WRAPPER fdw_name**
Specifies the name of the foreign data wrapper.
Value range: **fdw_name** is the data wrapper created by the system during database initialization. Currently, **fdw_name** can only be **gc_fdw** for other homogeneous clusters. You can also create **dist_fdw**, **file_fdw**, and **log_fdw**. **log_fdw** is used only for syntax compatibility and can be used to create foreign tables, but it is meaningless. **dist_fdw** is used to import GDS data. You do not need to manually create a server using **dist_fdw** because **gsmpp_server** is already built in.
- **OPTIONS ({ option_name ' value ' } [, ...])**
Specifies the parameters for the foreign server. The detailed parameter description is as follows:
 - address
Address of the foreign server.

- dbname
Database name of the foreign server.
- username
Username of a foreign server.
- password
Password of the foreign server.

Examples

Create **my_server**, in which **dfs_fdw** is the built-in foreign data wrapper.

```
-- Create my_server.  
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER file_fdw;  
  
-- Delete my_server.  
gaussdb=# DROP SERVER my_server;
```

You are advised to create another server in the homogeneous cluster, where **gc_fdw** is the foreign data wrapper in the database.

```
-- Create a server.  
gaussdb=# CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS  
  (address '10.146.187.231:8000,10.180.157.130:8000' ,  
   dbname 'test',  
   username 'test',  
   password '*****'  
  );  
  
-- Delete the server.  
gaussdb=# DROP SERVER server_remote;
```

Helpful Links

[ALTER SERVER](#) and [DROP SERVER](#)

7.14.74 CREATE SYNONYM

Description

Creates a synonym object. A synonym is an alias of a database object and is used to record the mapping between database object names. You can use synonyms to access associated database objects.

Precautions

- The user of a synonym should be its owner.
- If the schema name is specified, create a synonym in the specified schema. Otherwise create a synonym in the current schema.
- Database objects that can be accessed using synonyms include tables, views, types, packages, functions, and stored procedures.
- To use synonyms, you must have the required permissions on associated objects.
- The following DML statements support synonyms: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **EXPLAIN**, and **CALL**.

- The **CREATE SYNONYM** statement of an associated function or stored procedure cannot be used in a stored procedure. You are advised to use synonyms existing in the pg_synonym system catalog in the stored procedure.
- You are advised not to create synonyms for temporary tables. To create a synonym, you need to specify the schema name of the target temporary table. Otherwise, the synonym cannot be used normally. In addition, you need to run the **DROP SYNONYM** command before the current session ends.
- After an original object is deleted, the synonym associated with the object will not be deleted in cascading mode. If you continue to access the synonym, for tables, a message is displayed indicating that the synonyms have expired; for functions, stored procedures, and packages, a message is displayed indicating that the objects do not exist.
- Users granted the CREATE ANY SYNONYM permission can create synonyms in user schemas.
- Synonyms cannot be created for encrypted tables that contain encrypted columns and views, functions, and stored procedures based on encrypted tables.
- If the schema of a synonym is the schema to which the user belongs, the owner of the synonym is also the owner of the schema. In other scenarios, the owner of the synonym is the creator of the synonym by default.
- If **SEARCH_PATH** is set and no synonym schema is specified, for stored procedures and functions, the PG_PROC table is preferentially searched by name. If no function with the same name exists, synonyms are searched based on **SEARCH_PATH**. For other objects, **SEARCH_PATH** is preferentially searched, if their schemas are the same as that of synonyms, the objects are accessed prior to synonyms.
- Objects associated with synonyms cannot be accessed using DDL statements, such as CREATE, DROP, and ALTER.

Syntax

```
CREATE [ OR REPLACE ] SYNONYM synonym_name  
FOR object_name;
```

Parameters

- **OR REPLACE**
(Optional) Redefines the synonym if it already exists.
- **synonym_name**
Specifies the name of the synonym to be created, which can contain the schema name.
Value range: a string. It must comply with the [naming convention](#).
- **object_name**
Specifies the name of an object that is associated (optionally with schema names).
Value range: a string. It must comply with the [naming convention](#).

 NOTE

- **object_name** can be the name of an object that does not exist.
- **object_name** can be the name of a remote object accessed by using a database link. For details about how to use database links, see [DATABASE LINK](#).

 CAUTION

Do not create aliases for functions that contain passwords and other sensitive information, such as the encryption function `gs_encrypt` and the decryption function `gs_decrypt` or use aliases to call the functions to prevent sensitive information leakage.

Examples

```
-- Create schema ot.
gaussdb=# CREATE SCHEMA ot;

-- Create table ot.t1 and its synonym t1.
gaussdb=# CREATE TABLE ot.t1(id int, name varchar2(10)) DISTRIBUTE BY hash(id);
gaussdb=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

-- Use synonym t1.
gaussdb=# SELECT * FROM t1;
gaussdb=# INSERT INTO t1 VALUES (1, 'ada'), (2, 'bob');
gaussdb=# UPDATE t1 SET t1.name = 'cici' WHERE t1.id = 2;

-- Create synonym v1 and its associated view ot.v_t1.
gaussdb=# CREATE SYNONYM v1 FOR ot.v_t1;
gaussdb=# CREATE VIEW ot.v_t1 AS SELECT * FROM ot.t1;

-- Use synonym v1.
gaussdb=# SELECT * FROM v1;

-- Create overloaded function ot.add and its synonym add.
gaussdb=# CREATE OR REPLACE FUNCTION ot.add(a integer, b integer) RETURNS integer AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

gaussdb=# CREATE OR REPLACE FUNCTION ot.add(a decimal(5,2), b decimal(5,2)) RETURNS decimal(5,2)
AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

gaussdb=# CREATE OR REPLACE SYNONYM add FOR ot.add;

-- Use synonym add.
gaussdb=# SELECT add(1,2);
gaussdb=# SELECT add(1.2,2.3);

-- Create stored procedure ot.register and its synonym register.
gaussdb=# CREATE PROCEDURE ot.register(n_id integer, n_name varchar2(10))
SECURITY INVOKER
AS
BEGIN
INSERT INTO ot.t1 VALUES(n_id, n_name);
END;
/
```

```
gaussdb=# CREATE OR REPLACE SYNONYM register FOR ot.register;
-- Use synonym register to call the stored procedure.
gaussdb=# CALL register(3,'mia');

-- Delete the synonym.
gaussdb=# DROP SYNONYM t1;
gaussdb=# DROP SYNONYM IF EXISTS v1;
gaussdb=# DROP SYNONYM IF EXISTS add;
gaussdb=# DROP SYNONYM register;
gaussdb=# DROP SCHEMA ot CASCADE;
```

Helpful Links

[ALTER SYNONYM](#) and [DROP SYNONYM](#)

7.14.75 CREATE TABLE

Description

Creates an initially empty table in the current database. The table will be owned by the creator.

Precautions

- If a primary key constraint or unique constraint is added to a non-distribution key, a global secondary index is created by default.
- By default, **HASH(column_name)** is used in a distributed system. **column_name** is the primary key column or unique constraint column (if any) of the table, or the first column of a data type that can be used as the distribution key. The priority is as follows: primary key column > unique constraint column > first column of a data type that can be used as the distribution key. If a primary key column and multiple unique constraint columns exist and the distribution mode of the table is not specified, GSIs are created for other unique constraint columns by default after the distribution key of the table is determined based on the priority.
- Distribution columns do not support the UPDATE operation.
- If an error occurs during table creation, after it is fixed, the system may fail to delete the empty disk files created before the last automatic clearance. This problem seldom occurs and does not affect system running of the database.
- When JDBC is used, the **DEFAULT** value can be set through **PrepareStatement**.
- Row-store tables do not support foreign key as the table-level constraint.
- A user with the CREATE ANY TABLE permission can create tables in the public and user schemas. To create a table that contains serial columns, you must also obtain the CREATE ANY SEQUENCE permission to create sequences.
- The XML type cannot be used as a primary key or foreign key.

NOTICE

If GaussDB creates unlimited tables, CNs may be affected as follows:

- Resource exhaustion: Each table occupies certain disk space. Unlimited table creation will occupy a large amount of memory and disk space, which may exhaust CN resources. As a result, the system breaks down or becomes unstable.
- Performance deterioration: Unlimited table creation causes a large number of I/O operations and CPU computing, and the metadata information of the database becomes large, which may deteriorate the CN performance, including operations such as insert, query, update, and deletion. As a result, the system responds slowly or cannot meet service requirements.
- Security issues: Excessive tables make database management and maintenance difficult. Creating unlimited tables may cause security issues such as data leakage or data loss. Database stability decreases, causing immeasurable loss to enterprises.

Therefore, plan the number and size of GaussDB databases properly to avoid unlimited table creation and ensure system stability, reliability, and security.

- The number of table constraints cannot exceed 32,767.

Syntax

- Create a table.

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ]
table_name
    ( { column_name data_type [ CHARACTER SET | CHARSET charset ]
    [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option [...] ] }
    [ , ... ] )
[ WITH ( { storage_parameter = value } [ , ... ] ) ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
[ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY { REPLICATION | HASH ( column_name [ , ... ] )
| RANGE ( column_name [ , ... ] ) { SLICE REFERENCES tablename | ( slice_less_than_item [ , ... ] )
| ( slice_start_end_item [ , ... ] ) }
| LIST ( column_name [ , ... ] ) { SLICE REFERENCES tablename | ( slice_values_item [ , ... ] ) }
} ]
[ TO { GROUP groupname | NODE ( nodename [ , ... ] ) } ];
```

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
NULL |
CHECK ( expression ) |
DEFAULT default_expr |
GENERATED ALWAYS AS ( generation_expr ) [STORED] |
AUTO_INCREMENT |
UNIQUE [KEY] [ index_parameters ] |
PRIMARY KEY [ index_parameters ] |
ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY = column_encryption_key,
ENCRYPTION_TYPE = encryption_type_value ) }
REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
[ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **table_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
```

- ```

UNIQUE (column_name [, ...]) [index_parameters] |
PRIMARY KEY (column_name [, ...]) [index_parameters]
}
[DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE]

```
- **like\_option** is as follows:  
{ INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | PARTITION | REOPTIONS | DISTRIBUTION | ALL }
  - Range distribution rules  
For **slice\_less\_than\_item**:  
SLICE slice\_name VALUES LESS THAN ( { expression | MAXVALUE } [, ...] ) [ DATANODE datanode\_name | ( datanode\_name\_list ) ]  
For **slice\_start\_end\_item**:  
SLICE name {  
{ START ( expression ) END ( expression ) EVERY ( expression ) } |  
{ START ( literal ) END ( { literal | MAXVALUE } ) } |  
{ START ( literal ) } |  
{ END ( { literal | MAXVALUE } ) }  
}  
- The LIST distribution rule **slice\_values\_item** is as follows:  
SLICE name VALUES (expression [, ... ]) [DATANODE datanode\_name | ( datanode\_name\_list )] |  
SLICE name VALUES (DEFAULT) [DATANODE datanode\_name | ( datanode\_name\_list )]

**index\_parameters** is as follows:  
[ WITH ( {storage\_parameter = value} [, ... ] ) ]  
[ USING INDEX TABLESPACE tablespace\_name ]

## Parameters

- **UNLOGGED**

If this keyword is specified, the created table is an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, an unlogged table is automatically truncated after conflicts, OS restart, database restart, switchover, power-off, or abnormal shutdown, incurring data loss risks. Contents of an unlogged table are also not replicated to standby nodes. Any indexes created on an unlogged table are not automatically logged as well.

Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.

Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should rebuild indexes with errors.

- **GLOBAL | LOCAL**

When creating a temporary table, you can specify the **GLOBAL** or **LOCAL** keyword before **TEMP** or **TEMPORARY**. Currently, the two keywords are used to be compatible with the SQL standard. A local temporary table will be created by the GaussDB regardless of whether **GLOBAL** or **LOCAL** is specified.

- **TEMPORARY | TEMP**

If **TEMP** or **TEMPORARY** is specified, the created table is a temporary table. A temporary table is automatically dropped at the end of the current session. Therefore, you can create and use temporary tables in the current session as long as the connected CN in the session is normal. Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are advised not to perform operations on temporary tables in DDL statements. **TEMP** is equivalent to **TEMPORARY**.

---

**NOTICE**

- Temporary tables are visible to the current session through the schema starting with **pg\_temp** start. Users should not delete schema started with **pg\_temp** or **pg\_toast\_temp**.
- If **TEMPORARY** or **TEMP** is not specified when you create a table but its schema is set to that starting with **pg\_temp\_** in the current session, the table will be created as a temporary table.
- A temporary table is visible only to the current session. Therefore, it cannot be used together with **\parallel on**.
- Temporary tables do not support DN faults or primary/standby switchovers.

---

- **IF NOT EXISTS**

Sends a notice, but does not throw an error, if a table with the same name exists.

- **table\_name**

Specifies the name of the table to be created.

---

**NOTICE**

Some processing logic of materialized views determines whether a table is the log table of a materialized view or a table associated with a materialized view based on the table name prefix. Therefore, do not create a table whose name prefix is **mlog\_** or **matviewmap\_**. Otherwise, some functions of the table are affected.

---

- **column\_name**

Specifies the name of a column to be created in the new table.

- **data\_type**

Specifies the data type of the column.

- **compress\_mode**

Specifies whether to compress a table column. The option specifies the algorithm preferentially used by table columns. Row-store tables do not support compression.

Value range: **DELTA**, **PREFIX**, **DICTIONARY**, **NUMSTR**, and **NOCOMPRESS**

- **COLLATE collation**

Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **select \* from pg\_collation** command to query collation rules from the **pg\_collation** system catalog. The default collation rule is the row starting with **default** in the query result.

- **LIKE source\_table [ like\_option ... ]**

Specifies a table from which the new table automatically inherits all column names, their data types, and their not-null constraints, as well as the default expression declared as serial.

The new table and the source table are decoupled after creation is complete. Changes to the source table will not be applied to the new table, and it is not possible to include data of the new table in scans of the source table.

The copied columns and constraints are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another LIKE clause, an error is reported.

- The default expressions are copied from the source table to the new table only if **INCLUDING DEFAULTS** is specified. The **serial** column is not copied. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
- The CHECK constraints are copied from the source table to the new table only when **INCLUDING CONSTRAINTS** is specified. Other types of constraints are never copied to the new table. Not-null constraints are always copied to the new table. These rules also apply to column constraints and table constraints.
- Any indexes on the source table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
- **STORAGE** settings for the source column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- If **INCLUDING COMMENTS** is specified, comments for the columns, constraints, and indexes of the source table are copied. The default behavior is to exclude comments.
- If **INCLUDING PARTITION** is specified, the partition definitions of the source table are copied to the new table, and the new table no longer uses the PARTITION BY clause. The default behavior is to exclude partition definition of the source table.
- If **INCLUDING REOPTIONS** is specified, the new table will copy the storage parameter (that is, WITH clause) of the source table. The default behavior is to exclude partition definition of the storage parameter of the source table.
- If **INCLUDING DISTRIBUTION** is specified, the distribution information of the source table is copied to the new table, including distribution type and key, and the new table no longer use the **DISTRIBUTE BY** clause. The default behavior is to exclude distribution information of the source table.
- **INCLUDING ALL** contains the meaning of **INCLUDING DEFAULTS**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING PARTITION**, **INCLUDING REOPTIONS**, and **INCLUDING DISTRIBUTION**.

### NOTICE

- If the source table contains a sequence with the SERIAL, BIGSERIAL, or SMALLSERIAL data type, or a column in the source table is a sequence by default and the sequence is created for this table (created by using **CREATE SEQUENCE ... OWNED BY**), these sequences will not be copied to the new table, and another sequence specific to the new table will be created. This is different from earlier versions. To share a sequence between the source table and new table, create a shared sequence (do not use **OWNED BY**) and set a column in the source table to this sequence.
- You are advised not to set a column in the source table to the sequence specific to another table especially when the table is distributed in specific node groups, because doing so may result in CREATE TABLE ... LIKE execution failures. In addition, doing so may cause the sequence to become invalid in the source sequence because the sequence will also be deleted from the source table when it is deleted from the table that the sequence is specific to. To share a sequence among multiple tables, you are advised to create a shared sequence for them.

- **WITH ( { storage\_parameter = value } [, ... ] )**

Specifies an optional storage parameter for a table or an index. The WITH clause used for tables can also contain **OIDS=FALSE** to specify that rows of the new table should not contain OIDs.

### NOTE

When using the numeric type of arbitrary precision to define a column, specify precision **p** and scale **s**. When precision and scale are not specified, the input will be displayed.

The description of parameters is as follows:

- **FILLFACTOR**

The fill factor of a table is a percentage from 10 to 100. **100** (complete filling) is the default value. If the Ustore is used, the default value is **92**. When a smaller fill factor is specified, **INSERT** operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate.

Value range: 10–100.

- **ORIENTATION**

Specifies the storage mode of table data. This parameter can be set only once.

Value range:

- **ROW** indicates that table data is stored in rows.

Row store applies to the OLTP service, which has many interactive transactions. An interaction involves many columns in the table. Using row store can improve the efficiency.

Default value:

If an ordinary tablespace is specified, the default is **ROW**.

– STORAGE\_TYPE

Specifies the storage engine type. This parameter cannot be modified once it is set.

Value range:

- **USTORE** indicates that tables support the inplace-update storage engine. Note that the **track\_counts** and **track\_activities** parameters must be enabled when the Ustore table is used. Otherwise, space bloating may occur.
- **ASTORE** indicates that tables support the append-only storage engine.

Default value:

If no table is specified, data is stored in append-only mode by default.

– INIT\_TD

Specifies the number of TDs to be initialized when an Ustore table is created. This parameter can be modified by running the **ALTER TABLE** command. Note that this parameter affects the maximum size of a single tuple stored on the data page. The conversion method is  $MAX\_TUPLE\_SIZE = BLCKSZ - INIT\_TD * TD\_SIZE$ . For example, if you change the number of INIT\_TD from 4 to 8, the maximum size of a single tuple decreases by 4 x INIT\_TD.

Value ranges: 2–128. The default value is 4.

– segment

This parameter is reserved and is not supported currently.

– parallel\_workers

Number of bgworker threads started when an index is created. For example, value 2 indicates that two bgworker threads are started to create indexes concurrently.

Value range: [0,32], int type. If this parameter is not set, concurrent index creation is disabled.

Default value: If this parameter is not set, the concurrent index creation function is disabled.

– hasuids

If this parameter is set to **on**, a unique table-level ID is allocated to a tuple when the tuple is updated.

Value range: **on** and **off**.

Default value: **off**

– logical\_repl\_node

Name of the DN that returns logical logs to the CN during logical decoding of a distributed replication table. For the replication table, if this parameter is not specified, the first node in the node group where the current table is located is used by default. When the RESET operation is performed on this option, the node is reset to the first node of current table.

Value range: a string.

Default value: For the non-replication table, this parameter is empty by default. For the replication table, this parameter is set to the name of the first node by default.

- **WITHOUT OIDS**

It is equivalent to **WITH(OIDS=FALSE)**.

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS }**

**ON COMMIT** determines what to do when you commit a temporary table creation operation. Currently, the **PRESERVE ROWS** and **DELETE ROWS** options are supported.

- **PRESERVE ROWS** (default): No special action is taken at the ends of transactions. The temporary table and its table data are unchanged.
- **DELETE ROWS**: All rows in the temporary table will be deleted at the end of each transaction block.

- **COMPRESS | NOCOMPRESS**

If you specify **COMPRESS** in the **CREATE TABLE** statement, the compression feature is triggered in case of a bulk **INSERT** operation. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed. Row-store tables do not support compression.

Default value: **NOCOMPRESS**, that is, tuple data is not compressed before storage.

- **TABLESPACE tablespace\_name**

Specifies the tablespace where the new table is created. If it is not specified, the default tablespace is used.

- **DISTRIBUTE BY**

Specifies how the table is distributed or replicated between DNs.

Value range:

- **REPLICATION**: Each row in the table exists on all DNs, that is, each DN has complete table data.
- **HASH (column\_name)**: Each row of the table will be placed into specified DNs based on the hash value of the specified column.
- **RANGE(column\_name)**: maps a specified column based on the range and distributes data to the corresponding DNs.
- **LIST(column\_name)**: maps a specified column based on a specific value and distributes data to the corresponding DNs.

 NOTE

- For HASH distribution, the maximum number of distribution keys is the same as that of columns. A maximum of 1600 distribution keys are supported. For RANGE (VALUE LESS THAN) and LIST distributions, a maximum of four distribution key columns are supported. For RANGE (START END) distribution, only one distribution key column is supported.
- For a RANGE distribution policy using the VALUE LESS THAN clause, a maximum of four distribution key columns are supported. The distribution rules are as follows:
  1. The comparison starts from the first column of values to be inserted.
  2. If the value of the inserted first column is smaller than the boundary value of the current column in the local slice, the values are directly inserted.
  3. If the value of the inserted first column is equal to the boundary value of the current column in the local slice, compare the value of the inserted second column with the boundary value of the next column in the local slice. If the value of the inserted second column is smaller than the boundary value of the next column in the local slice, the values are directly inserted. If they are equal, the comparison of the next columns between the source and target continues.
  4. If the value of the inserted first column is greater than the boundary value of the current column in the local slice, compare the value with that in the next slice.
- If the shard of the RANGE distributed table corresponds to multiple DNs, the hash value of the distribution key is calculated to perform modulo operation on the number of DN and map a new DN. For details, see the example.
- If the shard of the list distributed table corresponds to multiple DNs, the hash value of the distribution key is calculated for the default shards. Then, use the calculated hash value to perform modulo operation on the number of DN and map a new DN. For non-default shards, the round robin method is used to map values in the values list to DNs. For details, see the example.
- The RANGE/LIST distribution tables support only scale-out but not scale-in. For details about the slice scale-out rule, contact the administrator.
- For a replication table, the following sufficient and necessary conditions must be met for pushing down:
  - Different types of window functions have different sufficient and necessary conditions for pushing down. There are four types:
    1. Unconditional pushdown is supported for functions such as the RANK series functions and RATIO\_TO\_REPORT functions. These window functions return consistent values for rows with the same PARTITION BY and ORDER BY values, regardless of the data relative sequence.
    2. Pushdown is supported when the projected columns exclude fields outside of PARTITION BY and ORDER BY or when PARTITION BY and ORDER BY form a primary key together, for example, window functions like ROW\_NUMBER and NTILE. Otherwise, when the projection column has the same value (but other column values are different), due to issues with relative sequence, the results returned by the window function on different DNs are different. As a result, data is inconsistent.
    3. Pushdown is supported when the value of **offset** is **0**. For the LAG and LEAD functions, when **offset** is **0**, a column is queried and does not depend on the relative data sequence. If the value of **offset** is greater than **0**, different tuples with the same value may be NULL due to different relative sequences.
    4. Pushdown is supported when the fields in the parameter expression are PARTITION BY columns, ORDER BY columns, or PARTITION BY

and ORDER BY columns that form a primary key, for example, functions such as FIRST\_VALUE, LAST\_VALUE, and NTH\_VALUE.

- If the query of the replication table contains system columns and volatile functions, the query is not pushed down.
- In the context of concatenating aggregate functions, the sufficient and necessary condition for pushing down is that the columns to be concatenated matches the ORDER BY columns, or the ORDER BY columns form a primary key.
- If the UPDATE or DELETE statement uses the WHERE CURRENT OF cursor\_name syntax, the query is not pushed down.

For the hash distribution, **column\_name** supports the following data types:

- Integer types: **TINYINT, SMALLINT, INT, BIGINT, and NUMERIC/DECIMAL**
- Character types: **CHAR, BPCHAR, VARCHAR, VARCHAR2, NVARCHAR2, and TEXT**
- Date/time types: **DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, INTERVAL, and SMALLDATETIME**

For the RANGE (VALUES LESS THAN) or LIST distribution, **column\_name** supports the following data types:

- Integer types: **SMALLINT, INT, BIGINT, NUMERIC/DECIMAL**
- Character types: **CHAR, BPCHAR, VARCHAR, VARCHAR2, NVARCHAR2, and TEXT**
- Date/Time types: **DATE, TIMESTAMP, TIMESTAMPTZ**

For the range (start end) distribution, the data type of **column\_name** must be one of the following:

- Integer types: **SMALLINT, INT, BIGINT, NUMERIC/DECIMAL**
- Date/Time types: **DATE, TIMESTAMP, TIMESTAMPTZ**

 NOTE

When you create a table, the choices of distribution keys and partition keys have major impact on SQL query performance. Therefore, select appropriate distribution keys and partition keys with strategies.

- Select appropriate distribution keys.

A hash table's distribution key should evenly distribute data on each DN to prevent skewing the data or distributing it unevenly across DNs. Determine appropriate distribution keys based on the following principles:

1. Determine whether data is skewed.

Connect to the database and run the following statement to check the number of tuples on each DN. Replace *tablename* with the actual name of the table to be analyzed.

```
gaussdb=# SELECT a.count,b.node_name FROM (SELECT count(*) AS
count,xc_node_id FROM tablename GROUP BY xc_node_id) a, pgxc_node b WHERE
a.xc_node_id=b.node_id ORDER BY a.count DESC;
```

If tuple numbers vary greatly (several times or tenfold) on each DN, a data skew occurs. Change the data distribution key based on the following principles:

2. Recreate a table to change its distribution keys. ALTER TABLE cannot change distribution keys. Therefore, you need to rebuild a table when changing its distribution keys.

Principles for selecting distribution keys are as follows:

The value of the distribution key should be discrete so that data can be evenly distributed on each DN. You can select the primary key of the table as the distribution key. For example, for a person information table, choose the ID card number column as the distribution key.

With the above principles are met, you can select join conditions as distribution keys so that join tasks can be pushed down to DNs, reducing the amount of data transferred between the DNs.

- Select appropriate partition keys.

In range partitioning, a table is partitioned based on ranges defined by one or more columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.

Modify partition keys to make the query result stored in the same or least partitions (partition pruning). Obtain consecutive I/O to improve the query performance.

In actual services, time is used to filter query objects. Therefore, you can use time as a partition key, and change the key value based on the total data volume and data volume of a single query.

- RANGE/LIST distribution

If no DN is specified for the shards of a RANGE/LIST distributed table, the database uses the Round Robin algorithm to allocate DNs to the shards. In addition, if RANGE/LIST distribution is used, you are advised to define as many shards as possible when creating a table for future capacity expansion. If the defined number of shards is less than the number of DNs before scale-out, data redistribution cannot be performed on new DNs. Note that the sharding rules are designed by users. In some extreme cases, scale-out may not solve the problem of insufficient storage space.

- **TO { GROUP *groupname* | NODE ( *nodename* [, ... ] ) }**

**TO GROUP** specifies the node group to which the table to be created belongs.  
**TO NODE** is used for internal scale-out tools.

- **CONSTRAINT *constraint\_name***

Specifies the name of a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an INSERT or UPDATE operation to succeed.

There are two ways to define constraints:

- A column constraint is defined as part of a column definition, and it is bound to a particular column.
- A table constraint is not bound to a particular column but can apply to more than one column.

- **NOT NULL**

Specifies that the column value cannot be **NULL**.

- **NULL**

Specifies that the column value can be **NULL**, which is the default value.

This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK ( expression )**

Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is true or unknown; otherwise, an error is thrown and the database is not altered.

A CHECK constraint specified as a column constraint should reference only the column's value, while an expression in a table constraint can reference multiple columns.

 **NOTE**

<>NULL and !=NULL are invalid in an expression. Change them to **IS NOT NULL**.

- **DEFAULT default\_expr**

Assigns a default data value to a column. The value can be any variable-free expressions. (Subqueries and cross-references to other columns in the current table are not allowed.) The data type of the default expression must match the data type of the column.

The default expression will be used in any INSERT operation that does not specify a value for the column. If there is no default value for a column, then the default value is **NULL**.

- **UNIQUE [KEY] index\_parameters**

**UNIQUE ( column\_name [, ... ] ) index\_parameters**

Specifies that a group of one or more columns of a table can contain only unique values.

For the purpose of a unique constraint, null is not considered equal.

UNIQUE KEY can be used only when **sql\_compatibility** is set to 'MYSQL', which has the same semantics as UNIQUE.

- **PRIMARY KEY index\_parameters**

**PRIMARY KEY ( column\_name [, ... ] ) index\_parameters**

Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-**NULL** values.

Only one primary key can be specified for a table.

- **REFERENCES**

The distributed database of the current version does not support the REFERENCES clause.

- **DEFERRABLE | NOT DEFERRABLE**

Determines whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only UNIQUE and PRIMARY KEY constraints accept this clause. All the other constraints are not deferrable.

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

If a constraint is deferrable, this clause specifies the default time to check the constraint.

- If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.
- If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

The constraint check time can be altered by executing the **SET CONSTRAINTS** command.

- **USING INDEX TABLESPACE tablespace\_name**

Specifies a tablespace in which an index associated with a UNIQUE or PRIMARY KEY constraint will be created. If this clause is not used, such index will be created in **default\_tablespace**. If **default\_tablespace** is empty, the default tablespace of the database is used.

- **ENCRYPTION\_TYPE = encryption\_type\_value**

For the encryption type in the ENCRYPTED WITH constraint, the value of **encryption\_type\_value** is **DETERMINISTIC** or **RANDOMIZED**.

## Examples

```
-- Create a simple table.
gaussdb=# CREATE TABLE tpcds.warehouse_t1
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) ,
 W_WAREHOUSE_SQ_FT INTEGER ,
 W_STREET_NUMBER CHAR(10) ,
 W_STREET_NAME VARCHAR(60) ,
 W_STREET_TYPE CHAR(15) ,
 W_SUITE_NUMBER CHAR(10) ,
 W_CITY VARCHAR(60) ,
 W_COUNTY VARCHAR(30) ,
 W_STATE CHAR(2) ,
 W_ZIP CHAR(10) ,
 W_COUNTRY VARCHAR(20) ,
 W_GMT_OFFSET DECIMAL(5,2) ,
);

gaussdb=# CREATE TABLE tpcds.warehouse_t2
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) ,
 W_WAREHOUSE_SQ_FT INTEGER ,
 W_STREET_NUMBER CHAR(10) ,
```

```
W_STREET_NAME VARCHAR(60) DICTIONARY,
W_STREET_TYPE CHAR(15)
W_SUITE_NUMBER CHAR(10)
W_CITY VARCHAR(60)
W_COUNTY VARCHAR(30)
W_STATE CHAR(2)
W_ZIP CHAR(10)
W_COUNTRY VARCHAR(20)
W_GMT_OFFSET DECIMAL(5,2)
);
-- Create a table and set the default value of the W_STATE column to GA.
gaussdb=# CREATE TABLE tpceds.warehouse_t3
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20)
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2) DEFAULT 'GA',
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
);
-- Create a table and check whether the W_WAREHOUSE_NAME column is unique at the end of its
creation.
gaussdb=# CREATE TABLE tpceds.warehouse_t4
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) UNIQUE DEFERRABLE,
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
);
-- Create a table with its fill factor set to 70%.
gaussdb=# CREATE TABLE tpceds.warehouse_t5
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20)
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2),
 UNIQUE(W_WAREHOUSE_NAME) WITH(fillfactor=70)
);
-- Alternatively, user the following syntax:
```

```
gaussdb=# CREATE TABLE tpceds.warehouse_t6
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) UNIQUE,
 W_WAREHOUSE_SQ_FT INTEGER ,
 W_STREET_NUMBER CHAR(10) ,
 W_STREET_NAME VARCHAR(60) ,
 W_STREET_TYPE CHAR(15) ,
 W_SUITE_NUMBER CHAR(10) ,
 W_CITY VARCHAR(60) ,
 W_COUNTY VARCHAR(30) ,
 W_STATE CHAR(2) ,
 W_ZIP CHAR(10) ,
 W_COUNTRY VARCHAR(20) ,
 W_GMT_OFFSET DECIMAL(5,2)
) WITH(fillfactor=70);

-- Create a table and specify that its data is not written to WALs.
gaussdb=# CREATE UNLOGGED TABLE tpceds.warehouse_t7
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) ,
 W_WAREHOUSE_SQ_FT INTEGER ,
 W_STREET_NUMBER CHAR(10) ,
 W_STREET_NAME VARCHAR(60) ,
 W_STREET_TYPE CHAR(15) ,
 W_SUITE_NUMBER CHAR(10) ,
 W_CITY VARCHAR(60) ,
 W_COUNTY VARCHAR(30) ,
 W_STATE CHAR(2) ,
 W_ZIP CHAR(10) ,
 W_COUNTRY VARCHAR(20) ,
 W_GMT_OFFSET DECIMAL(5,2)
);

-- Create a temporary table.
gaussdb=# CREATE TEMPORARY TABLE warehouse_t24
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) ,
 W_WAREHOUSE_SQ_FT INTEGER ,
 W_STREET_NUMBER CHAR(10) ,
 W_STREET_NAME VARCHAR(60) ,
 W_STREET_TYPE CHAR(15) ,
 W_SUITE_NUMBER CHAR(10) ,
 W_CITY VARCHAR(60) ,
 W_COUNTY VARCHAR(30) ,
 W_STATE CHAR(2) ,
 W_ZIP CHAR(10) ,
 W_COUNTRY VARCHAR(20) ,
 W_GMT_OFFSET DECIMAL(5,2)
);

-- Create a temporary table in a transaction and specify that this table is deleted when the transaction is
committed.
gaussdb=# CREATE TEMPORARY TABLE warehouse_t25
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) ,
 W_WAREHOUSE_SQ_FT INTEGER ,
 W_STREET_NUMBER CHAR(10) ,
 W_STREET_NAME VARCHAR(60) ,
 W_STREET_TYPE CHAR(15) ,
 W_SUITE_NUMBER CHAR(10) ,
 W_CITY VARCHAR(60) ,
```

```
W_COUNTY VARCHAR(30)
W_STATE CHAR(2)
W_ZIP CHAR(10)
W_COUNTRY VARCHAR(20)
W_GMT_OFFSET DECIMAL(5,2)
) ON COMMIT DELETE ROWS;

-- Create a table and specify that no error is reported for duplicate tables (if any).
gaussdb=# CREATE TABLE IF NOT EXISTS tpcds.warehouse_t8
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20)
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
);

-- Create an ordinary tablespace.
gaussdb=# CREATE TABLESPACE DS_TABLESPACE1 RELATIVE LOCATION 'tablespace/tablespace_1';
-- Specify a tablespace when creating a table.
gaussdb=# CREATE TABLE tpcds.warehouse_t9
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20)
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
) TABLESPACE DS_TABLESPACE1;

-- Separately specify the index tablespace for W_WAREHOUSE_NAME when creating the table.
gaussdb=# CREATE TABLE tpcds.warehouse_t10
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) UNIQUE USING INDEX TABLESPACE
DS_TABLESPACE1,
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
);
-- Create a table with a primary key constraint.
gaussdb=# CREATE TABLE tpcds.warehouse_t11
(
```

```
W_WAREHOUSE_SK INTEGER PRIMARY KEY,
W_WAREHOUSE_ID CHAR(16) NOT NULL,
W_WAREHOUSE_NAME VARCHAR(20)
W_WAREHOUSE_SQ_FT INTEGER
W_STREET_NUMBER CHAR(10)
W_STREET_NAME VARCHAR(60)
W_STREET_TYPE CHAR(15)
W_SUITE_NUMBER CHAR(10)
W_CITY VARCHAR(60)
W_COUNTY VARCHAR(30)
W_STATE CHAR(2)
W_ZIP CHAR(10)
W_COUNTRY VARCHAR(20)
W_GMT_OFFSET DECIMAL(5,2)
);

-- An alternative for the preceding syntax is as follows:
gaussdb=# CREATE TABLE tpcds.warehouse_t12
(
W_WAREHOUSE_SK INTEGER NOT NULL,
W_WAREHOUSE_ID CHAR(16) NOT NULL,
W_WAREHOUSE_NAME VARCHAR(20)
W_WAREHOUSE_SQ_FT INTEGER
W_STREET_NUMBER CHAR(10)
W_STREET_NAME VARCHAR(60)
W_STREET_TYPE CHAR(15)
W_SUITE_NUMBER CHAR(10)
W_CITY VARCHAR(60)
W_COUNTY VARCHAR(30)
W_STATE CHAR(2)
W_ZIP CHAR(10)
W_COUNTRY VARCHAR(20)
W_GMT_OFFSET DECIMAL(5,2),
PRIMARY KEY(W_WAREHOUSE_SK)
);

-- Or use the following statement to specify the name of the constraint:
gaussdb=# CREATE TABLE tpcds.warehouse_t13
(
W_WAREHOUSE_SK INTEGER NOT NULL,
W_WAREHOUSE_ID CHAR(16) NOT NULL,
W_WAREHOUSE_NAME VARCHAR(20)
W_WAREHOUSE_SQ_FT INTEGER
W_STREET_NUMBER CHAR(10)
W_STREET_NAME VARCHAR(60)
W_STREET_TYPE CHAR(15)
W_SUITE_NUMBER CHAR(10)
W_CITY VARCHAR(60)
W_COUNTY VARCHAR(30)
W_STATE CHAR(2)
W_ZIP CHAR(10)
W_COUNTRY VARCHAR(20)
W_GMT_OFFSET DECIMAL(5,2),
CONSTRAINT W_CSTR_KEY1 PRIMARY KEY(W_WAREHOUSE_SK)
);

-- Create a table with a compound primary key constraint.
gaussdb=# CREATE TABLE tpcds.warehouse_t14
(
W_WAREHOUSE_SK INTEGER NOT NULL,
W_WAREHOUSE_ID CHAR(16) NOT NULL,
W_WAREHOUSE_NAME VARCHAR(20)
W_WAREHOUSE_SQ_FT INTEGER
W_STREET_NUMBER CHAR(10)
W_STREET_NAME VARCHAR(60)
W_STREET_TYPE CHAR(15)
W_SUITE_NUMBER CHAR(10)
W_CITY VARCHAR(60)
W_COUNTY VARCHAR(30)
```

```
W_STATE CHAR(2)
W_ZIP CHAR(10)
W_COUNTRY VARCHAR(20)
W_GMT_OFFSET DECIMAL(5,2),
CONSTRAINT W_CSTR_KEY2 PRIMARY KEY(W_WAREHOUSE_SK, W_WAREHOUSE_ID)
);

-- Define a column check constraint.
gaussdb=# CREATE TABLE tpcds.warehouse_t19
(
 W_WAREHOUSE_SK INTEGER PRIMARY KEY CHECK (W_WAREHOUSE_SK > 0),
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) CHECK (W_WAREHOUSE_NAME IS NOT NULL),
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
);

gaussdb=# CREATE TABLE tpcds.warehouse_t20
(
 W_WAREHOUSE_SK INTEGER PRIMARY KEY,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20) CHECK (W_WAREHOUSE_NAME IS NOT NULL),
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2),
 CONSTRAINT W_CONSTR_KEY2 CHECK(W_WAREHOUSE_SK > 0 AND W_WAREHOUSE_NAME IS NOT
NULL)
);

-- Define a table with each row stored in all DN's.
gaussdb=# CREATE TABLE tpcds.warehouse_t21
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20)
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
)DISTRIBUTE BY REPLICATION;

Enable the primarynode option of the replication table.
```

```

gaussdb=# ALTER TABLE tpceds.warehouse_t21 SET (primarynode=on);

Check whether the option is enabled. (The content displayed in Options varies according to the version.)
gaussdb=# \d+ tpceds.warehouse_t21
 Table "tpceds.warehouse_t21"
 Column | Type | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
w_warehouse_sk | integer | | plain | |
w_warehouse_id | character(16) | not null | extended | |
w_warehouse_name | character varying(20) | | extended | |
w_warehouse_sq_ft | integer | | plain | |
w_street_number | character(10) | | extended | |
w_street_name | character varying(60) | | extended | |
w_street_type | character(15) | | extended | |
w_suite_number | character(10) | | extended | |
w_city | character varying(60) | | extended | |
w_county | character varying(30) | | extended | |
w_state | character(2) | | extended | |
w_zip | character(10) | | extended | |
w_country | character varying(20) | | extended | |
w_gmt_offset | numeric(5,2) | | main | |
Has OIDs: no
Distribute By: REPLICATION
Location Nodes: ALL DATANODES
Options: orientation=row, logical_repl_node=-1, compression=no, primarynode=on

-- Define a hash table.
gaussdb=# CREATE TABLE tpceds.warehouse_t22
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20)
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2),
 CONSTRAINT W_CONSTR_KEY3 UNIQUE(W_WAREHOUSE_SK)
)DISTRIBUTE BY HASH(W_WAREHOUSE_SK);

-- Define a table using RANGE distribution.
gaussdb=# CREATE TABLE tpceds.warehouse_t26
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20)
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
)DISTRIBUTE BY RANGE(W_WAREHOUSE_ID)
(
 SLICE s1 VALUES LESS THAN (10) DATANODE dn1,
 SLICE s2 VALUES LESS THAN (20) DATANODE dn2,
 SLICE s3 VALUES LESS THAN (30) DATANODE dn3,
 SLICE s4 VALUES LESS THAN (MAXVALUE) DATANODE dn4
)

```

```
);
-- Example of specifying multiple DNs for a shard in the range distributed table
gaussdb=# create table lrt_range (f_int1 int, f_int2 int, f_varchar1 varchar2(100))
distribute by range (f_int1, f_int2)
(
 slice s1 values less than (100, 100) datanode (datanode1,datanode2),
 slice s2 values less than (200, 200) datanode datanode2,
 slice s3 values less than (300, 300) datanode datanode2,
 slice s4 values less than (maxvalue, maxvalue) datanode (datanode1,datanode2)
);
-- Insert four data into shard s1.
gaussdb=# insert into lrt_range values(generate_series(1,4), generate_series(1,4));
gaussdb=# select node_name,node_type,node_id from pgxc_node;
 node_name | node_type | node_id
-----+-----+-----
coordinator1 | C | 1938253334
datanode1 | D | 888802358
datanode2 | D | -905831925
(3 rows)
-- View data distribution.
gaussdb=# select xc_node_id,* from lrt_range;
xc_node_id | f_int1 | f_int2 | f_varchar1
-----+-----+-----+-----
888802358 | 2 | 2 |
888802358 | 3 | 3 |
888802358 | 4 | 4 |
-905831925 | 1 | 1 |
(4 rows)
-- Map the DN based on the hash value for the shards in the range distributed table, insert (1,1) into
datanode2, and insert other data into datanode1.

-- Example of specifying multiple DNs for a shard in the list distributed table
gaussdb=# create table t_news
(
 county varchar(30),
 year varchar(60),
 name varchar(60),
 age int,
 news text
)distribute by list(county, year)
(
 slice s1 values (('china', '2020'),('china', '2021')) datanode (datanode1,datanode2),
 slice s2 values (('china', '2022'),('china', '2023'),('china', '2024')) datanode (datanode1,datanode2),
 slice s3 values (('china', '2025')) datanode (datanode1,datanode2),
 slice s4 values (('canada', '2021')) datanode datanode1,
 slice s5 values (('canada', '2022')) datanode datanode2,
 slice s6 values (('canada', '2023')) datanode datanode1,
 slice s7 values (('uk','2021')) datanode datanode1,
 slice s8 values (('uk','2022')) datanode datanode2,
 slice s9 values (('uk','2023')) datanode datanode1,
 slice s0 values (default) datanode (datanode1,datanode2)
);

-- Use the round robin mode to map DNs for the non-default shard.
-- For the shard s1, map ('china', '2020') to datanode1 and ('china', '2021') to datanode2.
-- For the shard s2, map ('china', '2022') to datanode1, ('china', '2023') to datanode2, and ('china', '2024')
to datanode1.
-- For the shard s3, map data ('china', '2025') to datanode1.

-- Delete data.
gaussdb=# delete from t_news;
DELETE 6
-- Use hash value to map DN for the default shard.
```

```
-- For the shard s0, map ('Japan', '2020') to datanode1 and other data to datanode2.

-- Example of multi-column range shard policy
gaussdb=# create table t_ran1(c1 int, c2 int, c3 int, c4 int, c5 int)
distribute by range(c1,c2)
(
SLICE s1 VALUES LESS THAN (10,10) DATANODE datanode1,
SLICE s2 VALUES LESS THAN (10,20) DATANODE datanode2,
SLICE s3 VALUES LESS THAN (20,10) DATANODE datanode3
);
gaussdb=# insert into t_ran1 values(9,5,'a');
gaussdb=# insert into t_ran1 values(9,20,'a');
gaussdb=# insert into t_ran1 values(9,21,'a');
gaussdb=# insert into t_ran1 values(10,5,'a');
gaussdb=# insert into t_ran1 values(10,15,'a');
gaussdb=# insert into t_ran1 values(10,20,'a');
gaussdb=# insert into t_ran1 values(10,21,'a');
gaussdb=# insert into t_ran1 values(11,5,'a');
gaussdb=# insert into t_ran1 values(11,20,'a');
gaussdb=# insert into t_ran1 values(11,21,'a');
gaussdb=# select node_name,node_type,node_id from pgxc_node;
 node_name | node_type | node_id
-----+-----+-----
coordinator1 | C | 1938253334
datanode1 | D | 888802358
datanode2 | D | -905831925
datanode3 | D | -1894792127
(4 rows)
gaussdb=# select xc_node_id,* from t_ran1;
xc_node_id | c1 | c2 | c3 | c4 | c5
-----+---+---+---+---+---
888802358 | 9 | 5 | 0 | |
888802358 | 9 | 20 | 0 | |
888802358 | 9 | 21 | 0 | |
888802358 | 10 | 5 | 0 | |
-905831925 | 10 | 15 | 0 | |
-1894792127 | 10 | 20 | 0 | |
-1894792127 | 10 | 21 | 0 | |
-1894792127 | 11 | 5 | 0 | |
-1894792127 | 11 | 20 | 0 | |
-1894792127 | 11 | 21 | 0 | |
(10 rows)

-- Create a table using SLICE REFERENCES.
gaussdb=# CREATE TABLE tpccs.warehouse_t27
(
W_WAREHOUSE_SK INTEGER NOT NULL,
W_WAREHOUSE_ID CHAR(16) NOT NULL,
W_WAREHOUSE_NAME VARCHAR(20) ,
W_WAREHOUSE_SQ_FT INTEGER ,
W_STREET_NUMBER CHAR(10) ,
W_STREET_NAME VARCHAR(60) ,
W_STREET_TYPE CHAR(15) ,
W_SUITE_NUMBER CHAR(10) ,
W_CITY VARCHAR(60) ,
W_COUNTY VARCHAR(30) ,
W_STATE CHAR(2) ,
W_ZIP CHAR(10) ,
W_COUNTRY VARCHAR(20) ,
W_GMT_OFFSET DECIMAL(5,2) ,
)DISTRIBUTE BY RANGE(W_WAREHOUSE_ID) SLICE REFERENCES warehouse_t26;

-- Define a table using LIST distribution.
gaussdb=# CREATE TABLE tpccs.warehouse_t28
(
W_WAREHOUSE_SK INTEGER NOT NULL,
W_WAREHOUSE_ID CHAR(16) NOT NULL,
W_WAREHOUSE_NAME VARCHAR(20) ,
W_WAREHOUSE_SQ_FT INTEGER ,
```

```
W_STREET_NUMBER CHAR(10)
W_STREET_NAME VARCHAR(60)
W_STREET_TYPE CHAR(15)
W_SUITE_NUMBER CHAR(10)
W_CITY VARCHAR(60)
W_COUNTY VARCHAR(30)
W_STATE CHAR(2)
W_ZIP CHAR(10)
W_COUNTRY VARCHAR(20)
W_GMT_OFFSET DECIMAL(5,2)
)DISTRIBUTE BY LIST(W_COUNTRY)
(
 SLICE s1 VALUES ('USA') DATANODE dn1,
 SLICE s2 VALUES ('CANADA') DATANODE dn2,
 SLICE s3 VALUES ('UK') DATANODE dn3,
 SLICE s4 VALUES (DEFAULT) DATANODE dn4
);
-- Add a varchar column to the tpcds.warehouse_t19 table.
gaussdb=# ALTER TABLE tpcds.warehouse_t19 ADD W_GOODS_CATEGORY varchar(30);

-- Add a check constraint to the tpcds.warehouse_t19 table.
gaussdb=# ALTER TABLE tpcds.warehouse_t19 ADD CONSTRAINT W_CONSTR_KEY4 CHECK (W_STATE IS NOT NULL);

-- Use one statement to alter the types of two existing columns.
gaussdb=# ALTER TABLE tpcds.warehouse_t19
 ALTER COLUMN W_GOODS_CATEGORY TYPE varchar(80),
 ALTER COLUMN W_STREET_NAME TYPE varchar(100);

-- This statement is equivalent to the preceding statement.
gaussdb=# ALTER TABLE tpcds.warehouse_t19 MODIFY (W_GOODS_CATEGORY varchar(30),
W_STREET_NAME varchar(60));

-- Add a not-null constraint to an existing column.
gaussdb=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY SET NOT NULL;

-- Remove not-null constraints from an existing column.
gaussdb=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY DROP NOT NULL;

-- Move a table to another tablespace.
gaussdb=# ALTER TABLE tpcds.warehouse_t19 SET TABLESPACE PG_DEFAULT;
-- Create the joe schema.
gaussdb=# CREATE SCHEMA joe;

-- Move a table to another schema.
gaussdb=# ALTER TABLE tpcds.warehouse_t19 SET SCHEMA joe;

-- Rename an existing table.
gaussdb=# ALTER TABLE joe.warehouse_t19 RENAME TO warehouse_t23;

-- Drop a column from the warehouse_t23 table.
gaussdb=# ALTER TABLE joe.warehouse_t23 DROP COLUMN W_STREET_NAME;

-- Create an encryption table.
gaussdb=# CREATE TABLE creditcard_info (id_number int, name text encrypted with
(column_encryption_key = lmgCEK, encryption_type = DETERMINISTIC), credit_card varchar(19) encrypted
with (column_encryption_key = lmgCEK1, encryption_type = DETERMINISTIC));
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'id_number' as the distribution column by
default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Drop the tablespace, schema joe, and schema tables warehouse.
gaussdb=# DROP TABLE tpcds.warehouse_t1;
gaussdb=# DROP TABLE tpcds.warehouse_t2;
gaussdb=# DROP TABLE tpcds.warehouse_t3;
gaussdb=# DROP TABLE tpcds.warehouse_t4;
```

```
gaussdb=# DROP TABLE tpcds.warehouse_t5;
gaussdb=# DROP TABLE tpcds.warehouse_t6;
gaussdb=# DROP TABLE tpcds.warehouse_t7;
gaussdb=# DROP TABLE tpcds.warehouse_t8;
gaussdb=# DROP TABLE tpcds.warehouse_t9;
gaussdb=# DROP TABLE tpcds.warehouse_t10;
gaussdb=# DROP TABLE tpcds.warehouse_t11;
gaussdb=# DROP TABLE tpcds.warehouse_t12;
gaussdb=# DROP TABLE tpcds.warehouse_t13;
gaussdb=# DROP TABLE tpcds.warehouse_t14;
gaussdb=# DROP TABLE tpcds.warehouse_t18;
gaussdb=# DROP TABLE tpcds.warehouse_t20;
gaussdb=# DROP TABLE tpcds.warehouse_t21;
gaussdb=# DROP TABLE tpcds.warehouse_t22;
gaussdb=# DROP TABLE joe.warehouse_t23;
gaussdb=# DROP TABLE tpcds.warehouse_t24;
gaussdb=# DROP TABLE tpcds.warehouse_t25;
gaussdb=# DROP TABLE tpcds.warehouse_t26;
gaussdb=# DROP TABLE tpcds.warehouse_t27;
gaussdb=# DROP TABLE tpcds.warehouse_t28;
gaussdb=# DROP TABLE creditcard_info;
gaussdb=# DROP TABLESPACE DS_TABLESPACE1;
gaussdb=# DROP SCHEMA IF EXISTS joe CASCADE;
```

## Helpful Links

[ALTER TABLE](#), [DROP TABLE](#), and [CREATE TABLESPACE](#)

## Suggestions

- UNLOGGED
  - The unlogged table and its indexes do not use the WAL mechanism during data writing. Their write speed is much higher than that of ordinary tables. Therefore, they can be used for storing intermediate result sets of complex queries to improve query performance.
  - The unlogged table has no primary/standby mechanism. In case of system faults or abnormal breakpoints, data loss may occur. Therefore, the unlogged table cannot be used to store basic data.
- TEMPORARY | TEMP
  - A temporary table is automatically dropped at the end of a session.
  - The temporary table is visible only to the current CN.
- LIKE
  - The new table automatically inherits all column names, data types, and NOT NULL constraints from this table. The new table is irrelevant to the source table after the creation.
- LIKE INCLUDING DEFAULTS
  - The default expressions are copied from the source table to the new table only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
- LIKE INCLUDING CONSTRAINTS
  - The CHECK constraints are copied from the source table to the new table only when **INCLUDING CONSTRAINTS** is specified. Other types of constraints are never copied to the new table. Not-null constraints are always copied to the new table. These rules also apply to column constraints and table constraints.

- LIKE INCLUDING INDEXES
  - Any indexes on the source table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
- LIKE INCLUDING STORAGE
  - **STORAGE** settings for the copied column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- LIKE INCLUDING COMMENTS
  - If **INCLUDING COMMENTS** is specified, comments for the columns, constraints, and indexes of the source table are copied. The default behavior is to exclude comments.
- LIKE INCLUDING PARTITION
  - If **INCLUDING PARTITION** is specified, the partition definitions of the source table are copied to the new table, and the new table no longer uses the PARTITION BY clause. The default behavior is to exclude partition definition of the source table.
- LIKE INCLUDING REOPTIONS
  - If **INCLUDING REOPTIONS** is specified, the new table will copy the storage parameter (that is, WITH clause) of the source table. The default behavior is to exclude partition definition of the storage parameter of the source table.
- LIKE INCLUDING DISTRIBUTION
  - If **INCLUDING DISTRIBUTION** is specified, the distribution information of the source table is copied to the new table, including distribution type and key, and the new table no longer use the DISTRIBUTE BY clause. The default behavior is to exclude distribution information of the source table.
- LIKE INCLUDING ALL
  - **INCLUDING ALL** contains the meaning of **INCLUDING DEFAULTS**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING PARTITION**, **INCLUDING REOPTIONS**, and **INCLUDING DISTRIBUTION**.
- ORIENTATION ROW
  - Creates a row-store table. Row-store applies to the OLTP service, which has many interactive transactions. An interaction involves many columns in the table. Using row-store can improve the efficiency.
- DISTRIBUTE BY
  - It is recommended that a fact table or dimension table containing a large amount of data be created as a distributed table. Each row of the table will be placed into specified DNs based on the hash value of the specified column. The syntax is **distribute by hash (column\_name)**.
  - It is recommended that a dimension table containing a small amount of data be created as a replication table. Each row in the table exists on all DNs. That is, each DN has complete table data. The syntax is **distribute by replication**.

## 7.14.76 CREATE TABLESPACE

### Description

Creates a tablespace in a database.

### Precautions

- A system administrator or a user who inherits permissions of the built-in role `gs_role_tablespace` can create a tablespace.
- Do not run **CREATE TABLESPACE** in a transaction block.
- If executing **CREATE TABLESPACE** fails but the internal directory (or file) has been created, the directory (or file) will remain. You need to manually clear it before creating the tablespace again. If there are residual files of soft links for the tablespace in the data directory, delete the residual files, and then perform O&M operations.
- **CREATE TABLESPACE** cannot be used for two-phase transactions. If it fails on some nodes, the execution cannot be rolled back.
- You are advised not to use user-defined tablespaces in scenarios such as Huawei Cloud.

This is because user-defined tablespaces are usually used with storage media other than the main storage (storage device where the default tablespace is located, such as a disk) to isolate I/O resources that can be used by different services. Storage devices use standard configurations and do not have other available storage media in scenarios such as Huawei Cloud. If the user-defined tablespace is not properly used, the system cannot run stably for a long time and the overall performance is affected. Therefore, you are advised to use the default tablespace.

### Syntax

```
CREATE TABLESPACE tablespace_name
 [OWNER user_name] [RELATIVE] LOCATION 'directory' [MAXSIZE 'space_size']
 [with_option_clause];
```

The **with\_option\_clause** syntax for creating a general tablespace is as follows:

```
WITH ({filesystem= { ' general ' | " general " | general } | address = { ' ip:port [, ...] ' | " ip:port [, ...] " } |
cfgpath = { ' path ' | " path " } | storepath = { ' rootpath ' | " rootpath " } | random_page_cost = { ' value ' | "
value " | value } | seq_page_cost = { ' value ' | " value " | value }}, ...)
```

### Parameters

- **tablespace\_name**  
Specifies name of a tablespace to be created.  
The tablespace name must be distinct from the name of any existing tablespace in the cluster and cannot start with "pg", which are reserved for system catalog spaces.  
Value range: a string. It must comply with the [naming convention](#).
- **OWNER user\_name**  
Specifies the name of the user who will own the tablespace. If omitted, the default owner is the current user.

Only system administrators can create tablespaces, but they can use the OWNER clause to assign ownership of tablespaces to non-system-administrators.

Value range: a string. It must be an existing user.

- **RELATIVE**

If this parameter is specified, a relative path is used. The location directory is relative to each CN/DN data directory.

Directory hierarchy: relative path of the CNs and DNs directory **/pg\_location/**. A relative path contains a maximum of two levels.

If this parameter is not specified, the absolute tablespace path is used. The location directory must be an absolute path.

- **LOCATION directory**

Specifies the directory for the table space. When creating an absolute tablespace path, ensure that the directory meets the following requirements:

- The GaussDB system user must have the read and write permissions on the directory, and the directory must be empty. If the directory does not exist, the system automatically creates it.
- The directory must be an absolute path, and does not contain special characters, such as dollar sign (\$) and greater-than sign (>).
- The directory cannot be specified under the database data directory.
- The directory must be a local path.

Value range: a string. It must be a valid directory.

- **MAXSIZE 'space\_size'**

Specifies the maximum value of the tablespace in a single DN.

Value range: a string consisting of a positive integer and unit. The unit can be KB, MB, GB, TB, or PB currently. The unit of parsed value is KB and cannot exceed the range that can be expressed in 64 bits, which is 1 KB to 9007199254740991 KB.

- **random\_page\_cost**

Specifies the cost of randomly reading the page overhead.

Value range: 0 to 1.79769e+308

Default value: value of the GUC parameter **random\_page\_cost**

- **seq\_page\_cost**

Specifies the cost of reading the page overhead in specified order.

Value range: 0 to 1.79769e+308

Default value: value of GUC parameter **seq\_page\_cost**

## Examples

```
-- Create a tablespace.
gaussdb=# CREATE TABLESPACE ds_location1 RELATIVE LOCATION 'test_tablespace/test_tablespace_1';

-- Create user joe.
gaussdb=# CREATE ROLE joe IDENTIFIED BY '*****';

-- Create user jay.
gaussdb=# CREATE ROLE jay IDENTIFIED BY '*****';

-- Create a tablespace and set its owner to user joe.
```

```
gaussdb=# CREATE TABLESPACE ds_location2 OWNER joe RELATIVE LOCATION 'test_tablespace/
test_tablespace_2';

-- Rename the ds_location1 tablespace to ds_location3.
gaussdb=# ALTER TABLESPACE ds_location1 RENAME TO ds_location3;

-- Change the owner of the ds_location2 tablespace.
gaussdb=# ALTER TABLESPACE ds_location2 OWNER TO jay;

-- Delete the tablespace.
gaussdb=# DROP TABLESPACE ds_location2;
gaussdb=# DROP TABLESPACE ds_location3;

-- Delete users.
gaussdb=# DROP ROLE joe;
gaussdb=# DROP ROLE jay;
```

## Helpful Links

[CREATE DATABASE](#), [CREATE TABLE](#), [CREATE INDEX](#), [DROP TABLESPACE](#), and [ALTER TABLESPACE](#)

## Suggestions

- create tablespace  
You are advised not to create tablespaces in a transaction.

## 7.14.77 CREATE TABLE AS

### Description

Creates a table from the results of a query.

CREATE TABLE AS creates a table and fills it with data obtained using SELECT. The table columns have the names and data types associated with the output columns of SELECT (except that you can override the SELECT output column names by giving an explicit list of new column names).

CREATE TABLE AS queries a source table once and writes the data in a new table. The result in the query view changes with the source table. In contrast, the view re-computes and defines its SELECT statement at each query.

### Precautions

- This statement cannot be used to create a partitioned table.
- If an error occurs during table creation, after it is fixed, the system may fail to delete the disk files that are created before the last automatic clearance and whose size is not 0. This problem seldom occurs and does not affect system running of the database.

### Syntax

```
CREATE [[GLOBAL | LOCAL] [TEMPORARY | TEMP] | UNLOGGED] TABLE [IF NOT EXISTS] table_name
 [(column_name [, ...])]
 [WITH ({storage_parameter = value} [, ...])]
 [COMPRESS | NOCOMPRESS]
 [TABLESPACE tablespace_name]
 [DISTRIBUTE BY { REPLICATION | HASH (column_name [, ...]) }
 | RANGE (column_name [, ...]) { SLICE REFERENCES tablename | (slice_less_than_item [, ...])
```

```
| (slice_start_end_item [, ...]) }
| LIST (column_name [, ...]) { SLICE REFERENCES tablename | (slice_values_item [, ...]) }
}]
[TO { GROUP groupname | NODE (nodename [, ...]) }]
AS query
[WITH [NO] DATA];
```

For details about each field, see [Syntax](#).

## Parameters

- **UNLOGGED**

Specifies that the table is created as an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, they are not crash-safe. An unlogged table is automatically truncated after a crash or unclean shutdown. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well.

- Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.
- Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should rebuild the indexes with errors.

- **GLOBAL | LOCAL**

When creating a temporary table, you can specify the GLOBAL or LOCAL keyword before TEMP or TEMPORARY. Currently, the two keywords are used to be compatible with the SQL standard. A local temporary table will be created by the GaussDB regardless of whether **GLOBAL** or **LOCAL** is specified.

- **TEMPORARY | TEMP**

If TEMP or TEMPORARY is specified, the created table is a temporary table. A temporary table is automatically dropped at the end of the current session. Therefore, you can create and use temporary tables in the current session as long as the connected CN in the session is normal. Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are advised not to perform operations on temporary tables in DDL statements. TEMP is equivalent to TEMPORARY.

---

### NOTICE

- Temporary tables are visible to the current session through the schema starting with **pg\_temp** start. Users should not delete schema started with **pg\_temp** or **pg\_toast\_temp**.
  - If TEMPORARY or TEMP is not specified when you create a table but its schema is set to that starting with **pg\_temp\_** in the current session, the table will be created as a temporary table.
  - A temporary table is visible only to the current session. Therefore, it cannot be used together with **\parallel on**.
  - Temporary tables do not support DN faults or primary/standby switchovers.
-

- **table\_name**  
Specifies the name of the table to be created.  
Value range: a string. It must comply with the [naming convention](#).
- **column\_name**  
Specifies the name of a column to be created in the new table.  
Value range: a string. It must comply with the [naming convention](#).
- **WITH ( storage\_parameter [= value] [, ... ] )**  
Specifies an optional storage parameter for a table or an index. See details of parameters below.
  - FILLFACTOR  
The fill factor of a table is a percentage from 10 to 100. **100** (complete filling) is the default value. If the Ustore is used, the default value is **92**. When a smaller fill factor is specified, INSERT operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate.  
Value range: 10–100
  - ORIENTATION  
Value range:  
**ROW** (default value): The data will be stored in rows.
  - COMPRESSION  
Specifies the compression level of table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time; and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution mode of table data loaded.  
Value range:  
Row-store tables do not support compression.
- **COMPRESS / NOCOMPRESS**  
Specifies keyword **COMPRESS** during the creation of a table, so that the compression feature is triggered in case of bulk **INSERT** operations. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed. Row-store tables do not support compression.  
Default value: **NOCOMPRESS**, that is, tuple data is not compressed before storage.
- **TABLESPACE tablespace\_name**  
Specifies that the new table will be created in the **tablespace\_name** tablespace. If not specified, the default tablespace is used.
- **DISTRIBUTE BY**  
For details, see [DISTRIBUTE BY](#).

- **TO { GROUP groupname | NODE ( nodename [, ... ] ) }**  
**TO GROUP** specifies the node group to which the table to be created belongs.  
**TO NODE** is used for internal scale-out tools.
- **AS query**  
Specifies a **SELECT** or **VALUES** command, or an **EXECUTE** command that runs a prepared **SELECT** or **VALUES** query.
- **[ WITH [ NO ] DATA ]**  
Specifies whether the data produced by the query should be copied to the new table. By default, the data will be copied. If the value **NO** is used, only the table structure will be copied.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.store_returns table.
gaussdb=# CREATE TABLE tpcds.store_returns
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 sr_item_sk VARCHAR(20) ,
 W_WAREHOUSE_SQ_FT INTEGER
);

-- Insert a record into a table:
gaussdb=# INSERT INTO tpcds.store_returns(W_WAREHOUSE_SK, W_WAREHOUSE_ID, sr_item_sk,
W_WAREHOUSE_SQ_FT) VALUES (1, 'AAAAAAAAABAAAAAAA', '4800', '20');

-- Create the tpcds.store_returns_t1 table and insert numbers that are greater than 4795 in the sr_item_sk
column of the tpcds.store_returns table.
gaussdb=# CREATE TABLE tpcds.store_returns_t1 AS SELECT * FROM tpcds.store_returns WHERE sr_item_sk
> '4795';

-- Copy tpcds.store_returns to create the tpcds.store_returns_t2 table.
gaussdb=# CREATE TABLE tpcds.store_returns_t2 AS table tpcds.store_returns;

-- Delete the table.
gaussdb=# DROP TABLE tpcds.store_returns_t1 ;
gaussdb=# DROP TABLE tpcds.store_returns_t2 ;
gaussdb=# DROP TABLE tpcds.store_returns;

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Helpful Links

[CREATE TABLE](#) and [SELECT](#)

## 7.14.78 CREATE TABLE PARTITION

### Description

Creates a partitioned table. Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and each physical piece is called a partition. Data is stored on these physical partitions, instead of the logical partitioned table.

The common forms of partitioning include range partitioning, interval partitioning, hash partitioning, list partitioning, and value partitioning. Currently, row-store tables support range partitioning, hash partitioning, list partitioning.

In range partitioning, a table is partitioned based on ranges defined by one or more columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.

The partitioning policy for range partitioning refers to how data is inserted into partitions.

In range partitioning, a table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned. Range partitioning is the most commonly used partitioning policy.

In hash partitioning, a modulus and a remainder are specified for each partition based on a column in the table, and records to be inserted into the table are allocated to the corresponding partition, the rows in each partition must meet the following condition: The value of the partition key divided by the specified modulus generates the remainder specified for the partition key.

In hash partitioning, table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

List partitioning is to allocate the records to be inserted into a table to the corresponding partition based on the key values in each partition. The key values do not overlap in different partitions. Create a partition for each group of key values to store corresponding data.

In list partitioning, table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

Partitioning can provide several benefits:

- Query performance can be improved drastically in certain situations, particularly when most of the heavily accessed rows of the table are in a single partition or a small number of partitions. Partitioning narrows the range of data search and improves data access efficiency.
- In the case of an insert or update operation on most portions of a single partition, performance can be improved by taking advantage of continuous scan of that partition instead of partitions scattered across the whole table.
- Frequent loading or deletion operations on records in a separate partition can be accomplished by reading or removing that partition. It also avoids the VACUUM overload caused by bulk DELETE operations.

## Precautions

- If the constraint key of the unique constraint and primary key constraint contains all partition keys, a local index is created for the constraints. Otherwise, a global index is created.
- Currently, hash partitioning supports only single-column partition keys, and does not support multi-column partition keys.

- In the PARTITION FOR (values) syntax for partitioned tables, values can only be constants.
- In the PARTITION FOR (values) syntax for partitioned tables, if data type conversion is required for values, you are advised to use forcible type conversion to prevent the implicit type conversion result from being inconsistent with the expected result.
- The maximum number of partitions is 1048575. Generally, it is impossible to create so many partitions, because too many partitions may cause insufficient memory. Create partitions based on the value of **local\_syscache\_threshold**. The memory used by the partitioned tables is about (number of partitions x 3/1024) MB. Theoretically, the memory occupied by the partitions cannot be greater than the value of **local\_syscache\_threshold**. In addition, some space must be reserved for other functions.
- If the memory is insufficient due to too many partitions, the performance deteriorates sharply.
- Currently, the statement specifying a partition cannot perform global index scan.
- Data of the XML type cannot be used as partition keys or level-2 partition keys.
- When UPDATE or DELETE is performed on a partitioned table, if the generated plan is not an FQS or Stream plan, the statement execution efficiency is low. You are advised to check statements and eliminate factors that cannot be pushed down to generate FQS or Stream plans.

## Syntax

```
CREATE TABLE [IF NOT EXISTS] partition_table_name
([
 { column_name data_type [COLLATE collation] [column_constraint [...]]
 | table_constraint
 | LIKE source_table [like_option [...]] }
 [, ...]
])
[WITH ({storage_parameter = value} [, ...])]
[COMPRESS | NOCOMPRESS]
[TABLESPACE tablespace_name]
[DISTRIBUTE BY { REPLICATION | HASH (column_name [, ...])
 | RANGE (column_name [, ...]) { SLICE REFERENCES tablename | (slice_less_than_item [, ...])
 | (slice_start_end_item [, ...]) }
 | LIST (column_name [, ...]) { SLICE REFERENCES tablename | (slice_values_item [, ...]) }
}]
[TO { GROUP groupname | NODE (nodename [, ...]) }]
PARTITION BY {
 {RANGE [COLUMNS] (partition_key) [PARTITIONS integer] (partition_less_than_item [, ...])} |
 {RANGE [COLUMNS] (partition_key) [PARTITIONS integer] (partition_start_end_item [, ...])} |
 {LIST [COLUMNS] (partition_key) [PARTITIONS integer] (PARTITION partition_name VALUES [IN]
(list_values) [TABLESPACE [=] tablespace_name][, ...])} |
 { HASH (partition_key) [PARTITIONS integer] (PARTITION partition_name [TABLESPACE [=]
tablespace_name][, ...])}
} [{ ENABLE | DISABLE } ROW MOVEMENT];
```

- **column\_constraint** is as follows:

```
[CONSTRAINT constraint_name]
{ NOT NULL |
 NULL |
 CHECK (expression) |
 DEFAULT default_expr |
 UNIQUE [KEY] [index_parameters] |
 PRIMARY KEY [index_parameters] }
[DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED | INITIALLY IMMEDIATE]
```

- **table\_constraint** is as follows:  
[ CONSTRAINT constraint\_name ]  
{ CHECK ( expression ) |  
UNIQUE ( column\_name [, ... ] ) [ index\_parameters ] |  
PRIMARY KEY ( column\_name [, ... ] ) [ index\_parameters ] }  
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
- **like\_option** is as follows:  
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS |  
REOPTIONS | DISTRIBUTION | ALL }
- **index\_parameters** is as follows:  
[ WITH ( {storage\_parameter = value} [, ... ] ) ]  
[ USING INDEX TABLESPACE tablespace\_name ]
- **partition\_less\_than\_item**:  
PARTITION partition\_name VALUES LESS THAN ( { partition\_value | MAXVALUE } ) [TABLESPACE  
tablespace\_name]
- **partition\_start\_end\_item**:  
PARTITION partition\_name {  
  {START(partition\_value) END (partition\_value) EVERY (interval\_value)} |  
  {START(partition\_value) END ({partition\_value | MAXVALUE})} |  
  {START(partition\_value)} |  
  {END({partition\_value | MAXVALUE})}  
} [TABLESPACE tablespace\_name]

## Parameters

- **IF NOT EXISTS**  
Sends a notice instead of throwing an error, if a table with the same name exists.
- **partition\_table\_name**  
Specifies the name of a partitioned table.  
Value range: a string. It must comply with the [naming convention](#).
- **column\_name**  
Specifies the name of a column to be created in the new table.  
Value range: a string. It must comply with the [naming convention](#).
- **data\_type**  
Specifies the data type of the column.
- **COLLATE collation**  
Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used.
- **CONSTRAINT constraint\_name**  
Specifies the name of a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an INSERT or UPDATE operation to succeed. You can run the **SELECT \* FROM pg\_collation** command to query collation rules from the pg\_collation system catalog. The default collation rule is the row starting with **default** in the query result.  
There are two ways to define constraints:
  - A column constraint is defined as part of a column definition, and it is bound to a particular column.
  - A table constraint is not bound to a particular column but can apply to more than one column.

- **LIKE source\_table [ like\_option ... ]**

Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

The target table and the source table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

Default expressions for the copied column definitions will be copied only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.

Not-null constraints are always copied to the new table. **CHECK** constraints will only be copied if **INCLUDING CONSTRAINTS** is specified; other types of constraints will never be copied. These rules also apply to column constraints and table constraints.

The copied columns and constraints are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.

- Any indexes on the original table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
- **STORAGE** settings for the original columns are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- If **INCLUDING COMMENTS** is specified, comments for the copied original columns, constraints, and indexes are copied. The default behavior is to exclude comments.
- If **INCLUDING REOPTIONS** is specified, the new table will copy the storage parameter (that is, **WITH** clause) of the original table. The default behavior is to exclude partition definition of the storage parameter of the original table.
- If **INCLUDING DISTRIBUTION** is specified, the new table will copy the distribution information of the original table, including distribution type and key, and the new table cannot use the **DISTRIBUTE BY** clause. The default behavior is to exclude distribution information of the original table.
- **INCLUDING ALL** contains the meaning of **INCLUDING DEFAULTS**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING REOPTIONS**, and **INCLUDING DISTRIBUTION**.

- **WITH ( storage\_parameter [= value] [, ... ] )**

Specifies an optional storage parameter for a table or an index. Optional parameters are as follows:

- **FILLFACTOR**

The fill factor of a table is a percentage from 10 to 100. **100** (complete filling) is the default value. When a smaller fill factor is specified, **INSERT** operations pack table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different

page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate.

Value range: 10–100

– **ORIENTATION**

Determines the data storage mode of the table.

Value range:

- **ROW** (default value): The data will be stored in rows.

---

**NOTICE**

**orientation** cannot be modified.

---

– **COMPRESSION**

- Row-store tables do not support compression.

• **COMPRESS / NOCOMPRESS**

Specifies keyword **COMPRESS** during the creation of a table, so that the compression feature is triggered in case of **BULK INSERT** operations. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed.

Default value: **NOCOMPRESS**, that is, tuple data is not compressed before storage. Row-store tables do not support compression.

• **TABLESPACE tablespace\_name**

Specifies that the new table will be created in the **tablespace\_name** tablespace. If not specified, the default tablespace is used.

• **DISTRIBUTE BY**

Specifies how the table is distributed or replicated between nodes.

For the value range and details, see [DISTRIBUTE BY](#).

• **TO { GROUP groupname | NODE ( nodename [, ... ] ) }**

**TO GROUP** specifies the node group to which the table to be created belongs. **TO NODE** is used for internal scale-out tools.

• **PARTITION BY RANGE [COLUMNS] (partition\_key)**

Creates a range partition. **partition\_key** is the name of the partition key.

The **COLUMNS** keyword can be used only when **sql\_compatibility** is set to **'MYSQL'**. The semantics of **PARTITION BY RANGE COLUMNS** is the same as that of **PARTITION BY RANGE**.

(1) Assume that the **VALUES LESS THAN** syntax is used.

---

**NOTICE**

If the **VALUE LESS THAN** clause is used, a range partitioning policy supports a partition key with up to 16 columns.

---

Data types supported by the partition keys are as follows: **SMALLINT**, **INTEGER**, **BIGINT**, **DECIMAL**, **NUMERIC**, **REAL**, **FLOAT4**, **FLOAT8**, **DOUBLE PRECISION**, **CHARACTER VARYING(*n*)**, **VARCHAR(*n*)**, **CHARACTER(*n*)**, **CHAR(*n*)**, **CHARACTER**, **CHAR**, **TEXT**, **NVARCHAR2**, **NAME**, **TIMESTAMP[(*p*)] [WITHOUT TIME ZONE]**, **TIMESTAMP[(*p*)] [WITH TIME ZONE]**, and **DATE**.

(2) Assume that the **START END** syntax is used.

---

#### NOTICE

In this case, only one partition key is supported.

---

Data types supported by the partition key are as follows: **SMALLINT**, **INTEGER**, **BIGINT**, **DECIMAL**, **NUMERIC**, **REAL**, **FLOAT4**, **FLOAT8**, **DOUBLE PRECISION**, **TIMESTAMP[(*p*)] [WITHOUT TIME ZONE]**, **TIMESTAMP[(*p*)] [WITH TIME ZONE]**, and **DATE**.

- **PARTITION partition\_name VALUES LESS THAN (( { partition\_value | MAXVALUE } [,...] ) | MAXVALUE }**

Specifies the information of partitions. **partition\_name** is the name of a range partition. **partition\_value** is the upper limit of a range partition, and the value depends on the type of **partition\_key**. **MAXVALUE** usually specifies the upper limit of the last range partition.

---

#### NOTICE

- Each partition requires an upper limit.
- The data type of the upper limit must be the same as that of the partition key.
- In a partition list, partitions are arranged in ascending order of upper limits. A partition with a smaller upper limit value is placed before another partition with a larger one.
- **MAXVALUE** that is not in parentheses can be used only when **sql\_compatibility** is set to 'MYSQL' and can have only one partition key.

- 
- **PARTITION partition\_name {START (partition\_value) END (partition\_value) EVERY (interval\_value)} | {START (partition\_value) END (partition\_value|MAXVALUE)} | {START(partition\_value)} | {END (partition\_value | MAXVALUE)}**

Specifies the information of partitions.

- **partition\_name**: name or name prefix of a range partition. It is the name prefix only in the following cases (assuming that **partition\_name** is **p1**):
  - If **START+END+EVERY** is used, the names of partitions will be defined as **p1\_1**, **p1\_2**, and the like. For example, if **PARTITION p1 START(1) END(4) EVERY(1)** is defined, the generated partitions are [1, 2), [2, 3), and [3, 4), and their names are **p1\_1**, **p1\_2**, and **p1\_3**. In this case, **p1** is a name prefix.
  - If the defined statement is in the first place and has **START** specified, the range (**MINVALUE**, **START**) will be automatically used as the first

actual partition, and its name will be **p1\_0**. The other partitions are then named **p1\_1**, **p1\_2**, and the like. For example, if **PARTITION p1 START(1), PARTITION p2 START(2)** is defined, generated partitions are (*MINVALUE*, 1), [1, 2), and [2, *MAXVALUE*), and their names will be **p1\_0**, **p1\_1**, and **p2**. In this case, **p1** is a name prefix and **p2** is a partition name. **MINVALUE** means the minimum value.

- **partition\_value**: start value or end value of a range partition. The value depends on **partition\_key** and cannot be *MAXVALUE*.
- **interval\_value**: width of each partition for dividing the [**START**, **END**) range. It cannot be *MAXVALUE*. If the value of (**END** - **START**) divided by **EVERY** has a remainder, the width of only the last partition is less than the value of **EVERY**.
- *MAXVALUE* usually specifies the upper limit of the last range partition.

---

#### NOTICE

1. If the defined statement is in the first place and has **START** specified, the range (*MINVALUE*, **START**) will be automatically used as the first actual partition.
2. The **START END** syntax must comply with the following rules:
  - The value of **START** (if any, same for the following situations) in each **partition\_start\_end\_item** must be smaller than that of **END**.
  - In two adjacent **partition\_start\_end\_item** statements, the value of the first **END** must be equal to that of the second **START**.
  - The value of **EVERY** in each **partition\_start\_end\_item** must be a positive number (in ascending order) and must be smaller than **END** minus **START**.
  - Each partition includes the start value (unless it is *MINVALUE*) and excludes the end value. The format is as follows: [**START**, **END**).
  - Partitions created by the same **partition\_start\_end\_item** belong to the same tablespace.
  - If **partition\_name** is a name prefix of a partition, the length must not exceed 57 bytes. If there are more than 57 bytes, the prefix will be automatically truncated.
  - When creating or modifying a partitioned table, ensure that the total number of partitions in the table does not exceed the maximum value **1048575**.
3. In statements for creating partitioned tables, **START END** and **LESS THAN** cannot be used together.
4. The **START END** syntax in a partitioned table creation SQL statement will be replaced by the **VALUES LESS THAN** syntax when **gs\_dump** is executed.

---

- **PARTITION BY LIST [COLUMNS] (partition\_key)**

Creates a list partition. **partition\_key** is the name of the partition key.

The **COLUMNS** keyword can be used only when **sql\_compatibility** is set to '**MYSQL**'. The semantics of **PARTITION BY LIST COLUMNS** is the same as that of **PARTITION BY LIST**.

- A list partitioning policy supports a partition key with up to 16 columns.
- For the clause syntax VALUES [IN] (list\_values), if **list\_values** contains the key values of the corresponding partition, it is recommended that the number of key values of each partition be less than or equal to 64.
- The clause VALUES IN can be used only when **sql\_compatibility** is set to 'MYSQL'. The semantics is the same as that of **VALUES**.

Partition keys support the following data types: INT1, INT2, INT4, INT8, NUMERIC, VARCHAR(*n*), CHAR, BPCHAR, NVARCHAR2, TIMESTAMP[(*p*)] [WITHOUT TIME ZONE], TIMESTAMP[(*p*)] [WITH TIME ZONE] and DATE. The number of partitions cannot exceed 1048575.

- **PARTITION BY HASH(partition\_key)**

Create a hash partition. **partition\_key** is the name of the partition key.

For **partition\_key**, the hash partitioning policy supports only one column of partition keys.

Partition keys support the following data types: INT1, INT2, INT4, INT8, NUMERIC, VARCHAR(*n*), CHAR, BPCHAR, TEXT, NVARCHAR2, TIMESTAMP[(*p*)] [WITHOUT TIME ZONE], TIMESTAMP[(*p*)] [WITH TIME ZONE], and DATE. The number of partitions cannot exceed 1048575.

- **PARTITIONS integer**

Specifies the number of partitions.

**integer** indicates the number of partitions. The value must be an integer greater than 0 and cannot be greater than 1048575.

- When this clause is specified after the range and list partitions, each partition must be explicitly defined, and the number of defined partitions must be equal to the integer value. This clause can be specified after the range and list partitions only when **sql\_compatibility** is set to 'MYSQL'.
- When this clause is specified after the hash and key partitions, if the definition of each partition is not listed, an *integer* number of partitions are automatically generated. The automatically generated partition name is "p+number", and the number ranges from 0 to *integer* minus 1. The tablespace of the partition is the tablespace of the table by default. If each partition definition is explicitly defined, the number of defined partitions must be the same as the value of *integer*. If neither the partition definition nor the number of partitions is specified, a unique partition is created.

- **{ ENABLE | DISABLE } ROW MOVEMENT**

Sets row movement.

If the tuple value is updated on the partition key during the UPDATE operation, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

Value range:

- **ENABLE**: Row movement is enabled.
- **DISABLE** (default value): Row movement is disabled.

If the row movement is enabled, an error may be reported when UPDATE and DELETE operations are performed concurrently. The causes are as follows:

The old data is marked as deleted in the UPDATE and DELETE operations. If the row movement is enabled, the cross-partition update occurs when the

partition key is updated. The kernel marks the old data in the old partition as deleted and adds a data to the new partition. As a result, the new data cannot be found by querying the old data.

If data in the same row is concurrently operated, the cross-partition and non-cross-partition data results have different behaviors in the following three concurrency scenarios: UPDATE and UPDATE concurrency, DELETE and DELETE concurrency, as well as UPDATE and DELETE concurrency.

- a. For non-cross-partition data, no error is reported for the second operation after the first operation is performed.

If the first operation is UPDATE, the latest data can be found and operated after the second operation is performed.

If the first operation is DELETE, the second operation is terminated if the current data is deleted and the latest data cannot be found.

- b. For the cross-partition data result, an error is reported for the second operation after the first operation is performed.

If the first operation is UPDATE, the second operation cannot find the latest data because the new data is in the new partition. Therefore, the second operation fails and an error is reported.

If the first operation is DELETE, performing the second operation can find that the current data is deleted and the latest data cannot be found, but cannot determine whether the operation of deleting the old data is UPDATE or DELETE. If the operation is UPDATE, an error is reported. If the operation is DELETE, the operation is terminated. To ensure the data correctness, an error is reported.

If the UPDATE and UPDATE concurrency, and UPDATE and DELETE concurrency are performed, the error can be solved only when the operations are performed serially. If the DELETE and DELETE concurrency are performed, the error can be solved by disabling the row movement.

- **NOT NULL**

The column is not allowed to contain null values. **ENABLE** can be omitted.

- **NULL**

Specifies that the column is allowed to contain null values. This is the default setting.

This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (condition) [ NO INHERIT ]**

Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is **TRUE** or **UNKNOWN**; otherwise, an error is thrown and the database is not altered.

A check constraint specified as a column constraint should reference only the column's values, while an expression in a table constraint can reference multiple columns.

A constraint marked with **NO INHERIT** will not propagate to child tables.

**ENABLE** can be omitted.

- **DEFAULT default\_expr**

Assigns a default data value to a column. The value can be any variable-free expressions. (Subqueries and cross-references to other columns in the current table are not allowed.) The data type of the default expression must match the data type of the column.

The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is null.

- **UNIQUE [KEY] index\_parameters**

**UNIQUE ( column\_name [, ... ] ) index\_parameters**

Specifies that a group of one or more columns of a table can contain only unique values.

For the purpose of a unique constraint, null is not considered equal.

UNIQUE KEY can be used only when **sql\_compatibility** is set to 'MYSQL', which has the same semantics as UNIQUE.

- **PRIMARY KEY index\_parameters**

**PRIMARY KEY ( column\_name [, ... ] ) index\_parameters**

Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

Only one primary key can be specified for a table.

- **DEFERRABLE | NOT DEFERRABLE**

Controls whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only **UNIQUE** and **PRIMARY KEY** constraints accept this clause. All the other constraints are not deferrable.

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

If a constraint is deferrable, this clause specifies the default time to check the constraint.

- If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.
- If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

The constraint check time can be altered using the **SET CONSTRAINTS** statement.

- **USING INDEX TABLESPACE tablespace\_name**

Allows selection of the tablespace in which the index associated with a **UNIQUE** or **PRIMARY KEY** constraint will be created. If not specified, the index is created in **default\_tablespace**. If **default\_tablespace** is empty, the default tablespace of the database is used.

## Examples

- Example 1: Create a range-partitioned table **tpcds.web\_returns\_p1**. The table has eight partitions and their partition keys are of the integer type. The ranges of the partitions are:  $wr\_returned\_date\_sk < 2450815$ ,  $2450815 \leq wr\_returned\_date\_sk < 2451179$ ,  $2451179 \leq wr\_returned\_date\_sk < 2451544$ ,  $2451544 \leq wr\_returned\_date\_sk < 2451910$ ,  $2451910 \leq wr\_returned\_date\_sk <$

2452275, 2452275 ≤ wr\_returned\_date\_sk < 2452640, 2452640 ≤  
wr\_returned\_date\_sk < 2453005, and wr\_returned\_date\_sk ≥ 2453005.

```
-- Create a temporary schema.
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# SET CURRENT_SCHEMA TO tpcds;

-- Create the tpcds.web_returns table.
gaussdb=# CREATE TABLE tpcds.web_returns
(
 W_WAREHOUSE_SK INTEGER NOT NULL,
 W_WAREHOUSE_ID CHAR(16) NOT NULL,
 W_WAREHOUSE_NAME VARCHAR(20)
 W_WAREHOUSE_SQ_FT INTEGER
 W_STREET_NUMBER CHAR(10)
 W_STREET_NAME VARCHAR(60)
 W_STREET_TYPE CHAR(15)
 W_SUITE_NUMBER CHAR(10)
 W_CITY VARCHAR(60)
 W_COUNTY VARCHAR(30)
 W_STATE CHAR(2)
 W_ZIP CHAR(10)
 W_COUNTRY VARCHAR(20)
 W_GMT_OFFSET DECIMAL(5,2)
);

-- Create a range-partitioned table tpcds.web_returns_p1.
gaussdb=# CREATE TABLE tpcds.web_returns_p1
(
 WR_RETURNED_DATE_SK INTEGER
 WR_RETURNED_TIME_SK INTEGER
 WR_ITEM_SK INTEGER NOT NULL,
 WR_REFUNDED_CUSTOMER_SK INTEGER
 WR_REFUNDED_CDEMO_SK INTEGER
 WR_REFUNDED_HDEMO_SK INTEGER
 WR_REFUNDED_ADDR_SK INTEGER
 WR_RETURNING_CUSTOMER_SK INTEGER
 WR_RETURNING_CDEMO_SK INTEGER
 WR_RETURNING_HDEMO_SK INTEGER
 WR_RETURNING_ADDR_SK INTEGER
 WR_WEB_PAGE_SK INTEGER
 WR_REASON_SK INTEGER
 WR_ORDER_NUMBER BIGINT NOT NULL,
 WR_RETURN_QUANTITY INTEGER
 WR_RETURN_AMT DECIMAL(7,2)
 WR_RETURN_TAX DECIMAL(7,2)
 WR_RETURN_AMT_INC_TAX DECIMAL(7,2)
 WR_FEE DECIMAL(7,2)
 WR_RETURN_SHIP_COST DECIMAL(7,2)
 WR_REFUNDED_CASH DECIMAL(7,2)
 WR_REVERSED_CHARGE DECIMAL(7,2)
 WR_ACCOUNT_CREDIT DECIMAL(7,2)
 WR_NET_LOSS DECIMAL(7,2)
)
DISTRIBUTE BY HASH (WR_ITEM_SK)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
 PARTITION P1 VALUES LESS THAN(2450815),
 PARTITION P2 VALUES LESS THAN(2451179),
 PARTITION P3 VALUES LESS THAN(2451544),
 PARTITION P4 VALUES LESS THAN(2451910),
 PARTITION P5 VALUES LESS THAN(2452275),
 PARTITION P6 VALUES LESS THAN(2452640),
 PARTITION P7 VALUES LESS THAN(2453005),
 PARTITION P8 VALUES LESS THAN(MAXVALUE)
);

-- Import data from the example data table.
gaussdb=# INSERT INTO tpcds.web_returns_p1 SELECT * FROM tpcds.web_returns;

-- Delete the P8 partition.
```

```
gaussdb=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION P8;

-- Add a partition WR_RETURNED_DATE_SK with values ranging from 2453005 to 2453105.
gaussdb=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P8 VALUES LESS THAN (2453105);

-- Add a partition WR_RETURNED_DATE_SK with values ranging from 2453105 to MAXVALUE.
gaussdb=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P9 VALUES LESS THAN (MAXVALUE);

-- Delete the P8 partition.
gaussdb=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION FOR (2453005);

-- Rename the P7 partition to P10.
gaussdb=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION P7 TO P10;

-- Rename the P6 partition to P11.
gaussdb=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION FOR (2452639) TO P11;

-- Query the number of rows in the P10 partition.
gaussdb=# SELECT count(*) FROM tpcds.web_returns_p1 PARTITION (P10);
count

0
(1 row)

-- Query the number of rows in the P1 partition.
gaussdb=# SELECT COUNT(*) FROM tpcds.web_returns_p1 PARTITION FOR (2450815);
count

0
(1 row)

-- Delete the tpcds.web_returns_p1 table.
gaussdb=# DROP TABLE tpcds.web_returns_p1;

-- Delete the tpcds.web_returns table.
gaussdb=# DROP TABLE tpcds.web_returns;

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

- Example 2: Create a range-partitioned table **tpcds.web\_returns\_p2**. The table has eight partitions and their partition keys are of the integer type. The upper limit of the eighth partition is *MAXVALUE*.

The ranges of the partitions are:  $wr\_returned\_date\_sk < 2450815$ ,  $2450815 \leq wr\_returned\_date\_sk < 2451179$ ,  $2451179 \leq wr\_returned\_date\_sk < 2451544$ ,  $2451544 \leq wr\_returned\_date\_sk < 2451910$ ,  $2451910 \leq wr\_returned\_date\_sk < 2452275$ ,  $2452275 \leq wr\_returned\_date\_sk < 2452640$ ,  $2452640 \leq wr\_returned\_date\_sk < 2453005$ , and  $wr\_returned\_date\_sk \geq 2453005$ .

The tablespace of the **tpcds.web\_returns\_p2** partitioned table is **example1**. Partitions **P1** to **P7** have no specified tablespaces, and use the **example1** tablespace of the **tpcds.web\_returns\_p2** partitioned table. The tablespace of the **P8** partitioned table is **example2**.

Assume that *CN and DN data directory/pg\_location/mount1/path1*, *CN and DN data directory/pg\_location/mount2/path2*, *CN and DN data directory/pg\_location/mount3/path3*, and *CN and DN data directory/pg\_location/mount4/path4* are empty directories for which user **dwsadmin** has read and write permissions.

```
gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
gaussdb=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
gaussdb=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
gaussdb=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';

-- Create a temporary schema.
gaussdb=# CREATE SCHEMA tpcds;
```

```
gaussdb=# SET CURRENT_SCHEMA TO tpcds;

gaussdb=# CREATE TABLE tpcds.web_returns_p2
(
 WR_RETURNED_DATE_SK INTEGER ,
 WR_RETURNED_TIME_SK INTEGER ,
 WR_ITEM_SK INTEGER NOT NULL,
 WR_REFUNDED_CUSTOMER_SK INTEGER ,
 WR_REFUNDED_CDEMO_SK INTEGER ,
 WR_REFUNDED_HDEMO_SK INTEGER ,
 WR_REFUNDED_ADDR_SK INTEGER ,
 WR_RETURNING_CUSTOMER_SK INTEGER ,
 WR_RETURNING_CDEMO_SK INTEGER ,
 WR_RETURNING_HDEMO_SK INTEGER ,
 WR_RETURNING_ADDR_SK INTEGER ,
 WR_WEB_PAGE_SK INTEGER ,
 WR_REASON_SK INTEGER ,
 WR_ORDER_NUMBER BIGINT NOT NULL,
 WR_RETURN_QUANTITY INTEGER ,
 WR_RETURN_AMT DECIMAL(7,2) ,
 WR_RETURN_TAX DECIMAL(7,2) ,
 WR_RETURN_AMT_INC_TAX DECIMAL(7,2) ,
 WR_FEE DECIMAL(7,2) ,
 WR_RETURN_SHIP_COST DECIMAL(7,2) ,
 WR_REFUNDED_CASH DECIMAL(7,2) ,
 WR_REVERSED_CHARGE DECIMAL(7,2) ,
 WR_ACCOUNT_CREDIT DECIMAL(7,2) ,
 WR_NET_LOSS DECIMAL(7,2)
)
TABLESPACE example1
DISTRIBUTE BY HASH (WR_ITEM_SK)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
 PARTITION P1 VALUES LESS THAN(2450815),
 PARTITION P2 VALUES LESS THAN(2451179),
 PARTITION P3 VALUES LESS THAN(2451544),
 PARTITION P4 VALUES LESS THAN(2451910),
 PARTITION P5 VALUES LESS THAN(2452275),
 PARTITION P6 VALUES LESS THAN(2452640),
 PARTITION P7 VALUES LESS THAN(2453005),
 PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;

-- Create a partitioned table using LIKE.
gaussdb=# CREATE TABLE tpcds.web_returns_p3 (LIKE tpcds.web_returns_p2 INCLUDING PARTITION);

-- Change the tablespace of the P1 partition to example2.
gaussdb=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P1 TABLESPACE example2;

-- Change the tablespace of the P2 partition to example3.
gaussdb=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P2 TABLESPACE example3;

-- Split the P8 partition at 2453010.
gaussdb=# ALTER TABLE tpcds.web_returns_p2 SPLIT PARTITION P8 AT (2453010) INTO
(
 PARTITION P9,
 PARTITION P10
);

-- Merge the P6 and P7 partitions into one.
gaussdb=# ALTER TABLE tpcds.web_returns_p2 MERGE PARTITIONS P6, P7 INTO PARTITION P8;

-- Modify the migration attribute of the partitioned table.
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;

-- Delete tables and tablespaces.
gaussdb=# DROP TABLE tpcds.web_returns_p1;
gaussdb=# DROP TABLE tpcds.web_returns_p2;
```

```

gaussdb=# DROP TABLE tpceds.web_returns_p3;
gaussdb=# DROP SCHEMA tpceds CASCADE;
gaussdb=# DROP TABLESPACE example1;
gaussdb=# DROP TABLESPACE example2;
gaussdb=# DROP TABLESPACE example3;
gaussdb=# DROP TABLESPACE example4;

```

- Example 3: Use **START END** to create and modify a range-partitioned table.

Assume that **/home/omm/startend\_tbs1**, **/home/omm/startend\_tbs2**, **/home/omm/startend\_tbs3**, and **/home/omm/startend\_tbs4** are empty directories for which user **omm** has the read and write permissions.

```

-- Create a tablespace.
gaussdb=# CREATE TABLESPACE startend_tbs1 LOCATION '/home/omm/startend_tbs1';
gaussdb=# CREATE TABLESPACE startend_tbs2 LOCATION '/home/omm/startend_tbs2';
gaussdb=# CREATE TABLESPACE startend_tbs3 LOCATION '/home/omm/startend_tbs3';
gaussdb=# CREATE TABLESPACE startend_tbs4 LOCATION '/home/omm/startend_tbs4';

-- Create a temporary schema.
gaussdb=# CREATE SCHEMA tpceds;
gaussdb=# SET CURRENT_SCHEMA TO tpceds;

-- Create a partitioned table with the partition key of the integer type.
gaussdb=# CREATE TABLE tpceds.startend_pt (c1 INT, c2 INT)
TABLESPACE startend_tbs1
DISTRIBUTE BY HASH (c1)
PARTITION BY RANGE (c2) (
PARTITION p1 START(1) END(1000) EVERY(200) TABLESPACE startend_tbs2,
PARTITION p2 END(2000),
PARTITION p3 START(2000) END(2500) TABLESPACE startend_tbs3,
PARTITION p4 START(2500),
PARTITION p5 START(3000) END(5000) EVERY(1000) TABLESPACE startend_tbs4
)
ENABLE ROW MOVEMENT;

-- View the information of the partitioned table.
gaussdb=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpceds.startend_pt'::regclass ORDER BY 1;
 relname | boundaries | spcname
-----+-----+-----
p1_0 | {1} | startend_tbs2
p1_1 | {201} | startend_tbs2
p1_2 | {401} | startend_tbs2
p1_3 | {601} | startend_tbs2
p1_4 | {801} | startend_tbs2
p1_5 | {1000} | startend_tbs2
p2 | {2000} | startend_tbs1
p3 | {2500} | startend_tbs3
p4 | {3000} | startend_tbs1
p5_1 | {4000} | startend_tbs4
p5_2 | {5000} | startend_tbs4
startend_pt | | startend_tbs1
(12 rows)

-- Import data and check the data volume in a partition.
gaussdb=# INSERT INTO tpceds.startend_pt VALUES (GENERATE_SERIES(0, 4999),
GENERATE_SERIES(0, 4999));
gaussdb=# SELECT COUNT(*) FROM tpceds.startend_pt PARTITION FOR (0);
 count

1
(1 row)

gaussdb=# SELECT COUNT(*) FROM tpceds.startend_pt PARTITION (p3);
 count

500
(1 row)

```

```

-- Add partitions [5000, 5300), [5300, 5600), [5600, 5900), and [5900, 6000).
gaussdb=# ALTER TABLE tpcds.startend_pt ADD PARTITION p6 START(5000) END(6000) EVERY(300)
TABLESPACE startend_tbs4;

-- Add the partition p7, specified by MAXVALUE.
gaussdb=# ALTER TABLE tpcds.startend_pt ADD PARTITION p7 END(MAXVALUE);

-- Rename the partition p7 to p8.
gaussdb=# ALTER TABLE tpcds.startend_pt RENAME PARTITION p7 TO p8;

-- Delete the partition p8.
gaussdb=# ALTER TABLE tpcds.startend_pt DROP PARTITION p8;

-- Rename the partition where 5950 is located to p71.
gaussdb=# ALTER TABLE tpcds.startend_pt RENAME PARTITION FOR(5950) TO p71;

-- Split the partition [4000, 5000) where 4500 is located.
gaussdb=# ALTER TABLE tpcds.startend_pt SPLIT PARTITION FOR(4500) INTO(PARTITION q1
START(4000) END(5000) EVERY(250) TABLESPACE startend_tbs3);

-- Change the tablespace of the partition p2 to startend_tbs4.
gaussdb=# ALTER TABLE tpcds.startend_pt MOVE PARTITION p2 TABLESPACE startend_tbs4;

-- View the partition status.
gaussdb=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
 relname | boundaries | spcname
-----+-----+-----
p1_0 | {1} | startend_tbs2
p1_1 | {201} | startend_tbs2
p1_2 | {401} | startend_tbs2
p1_3 | {601} | startend_tbs2
p1_4 | {801} | startend_tbs2
p1_5 | {1000} | startend_tbs2
p2 | {2000} | startend_tbs4
p3 | {2500} | startend_tbs3
p4 | {3000} | startend_tbs1
p5_1 | {4000} | startend_tbs4
p6_1 | {5300} | startend_tbs4
p6_2 | {5600} | startend_tbs4
p6_3 | {5900} | startend_tbs4
p71 | {6000} | startend_tbs4
q1_1 | {4250} | startend_tbs3
q1_2 | {4500} | startend_tbs3
q1_3 | {4750} | startend_tbs3
q1_4 | {5000} | startend_tbs3
startend_pt | | startend_tbs1
(19 rows)

-- Delete tables and tablespaces.
gaussdb=# DROP TABLE tpcds.startend_pt;
gaussdb=# DROP SCHEMA tpcds CASCADE;
gaussdb=# DROP TABLESPACE startend_tbs1;
gaussdb=# DROP TABLESPACE startend_tbs2;
gaussdb=# DROP TABLESPACE startend_tbs3;
gaussdb=# DROP TABLESPACE startend_tbs4;

```

- Example 4: Create list partitioned table **test\_list**. The table initially contains four partitions and the partition key is of the INT type. The ranges of the four partitions are 2000, 3000, 4000, and 5000 respectively.

```

-- Create the test_list table.
gaussdb=# CREATE TABLE test_list (col1 int, col2 int)
partition by list(col1)
(
partition p1 values (2000),
partition p2 values (3000),
partition p3 values (4000),
partition p4 values (5000)
);

```

```
-- Insert data.
gaussdb=# INSERT INTO test_list VALUES(2000, 2000);
INSERT 0 1
gaussdb=# INSERT INTO test_list VALUES(3000, 3000);
INSERT 0 1

-- View the partition information.
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
-----+-----+-----
p1 | l | {2000}
p2 | l | {3000}
p3 | l | {4000}
p4 | l | {5000}
(4 rows)

-- The inserted data does not match the partition, and an error is reported.
gaussdb=# INSERT INTO test_list VALUES(6000, 6000);
ERROR: inserted partition key does not map to any table partition

-- Add a partition.
gaussdb=# ALTER TABLE test_list add partition p5 values (6000);
ALTER TABLE
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
-----+-----+-----
p5 | l | {6000}
p4 | l | {5000}
p1 | l | {2000}
p2 | l | {3000}
p3 | l | {4000}
(5 rows)
gaussdb=# INSERT INTO test_list VALUES(6000, 6000);
INSERT 0 1

-- Exchange data between the partitioned table and ordinary table.
gaussdb=# CREATE TABLE t1 (col1 int, col2 int);
CREATE TABLE
gaussdb=# SELECT * FROM test_list partition (p1);
 col1 | col2
-----+-----
2000 | 2000
(1 row)
gaussdb=# ALTER TABLE test_list exchange partition (p1) with table t1;
ALTER TABLE
gaussdb=# SELECT * FROM test_list partition (p1);
 col1 | col2
-----+-----
(0 rows)
gaussdb=# SELECT * FROM t1;
 col1 | col2
-----+-----
2000 | 2000
(1 row)

-- Truncate the partition.
gaussdb=# SELECT * FROM test_list partition (p2);
 col1 | col2
-----+-----
3000 | 3000
(1 row)
gaussdb=# ALTER TABLE test_list truncate partition p2;
ALTER TABLE
gaussdb=# SELECT * FROM test_list partition (p2);
 col1 | col2
-----+-----
```

```

(0 rows)

-- Delete the partition.
gaussdb=# alter table test_list drop partition p5;
ALTER TABLE
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
-----+-----+-----
p4 | l | {5000}
p1 | l | {2000}
p2 | l | {3000}
p3 | l | {4000}
(4 rows)

gaussdb=# INSERT INTO test_list VALUES(6000, 6000);
ERROR: inserted partition key does not map to any table partition

-- Merge partitions.
gaussdb=# alter table test_list merge partitions p1,p2 into partition p2;
ALTER TABLE
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
-----+-----+-----
p2 | l | {2000,3000}
p4 | l | {5000}
p3 | l | {4000}
(3 rows)

-- Split partitions.
gaussdb=# alter table test_list split partition p2 values(2000) into (partition p1, partition p2);
ALTER TABLE
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
-----+-----+-----
p2 | l | {3000}
p1 | l | {2000}
p4 | l | {5000}
p3 | l | {4000}
(4 rows)

-- Delete the ordinary table.
gaussdb=# DROP TABLE t1;

-- Drop the partitioned table.
gaussdb=# drop table test_list;

```

- **Example 5: Create a hash partitioned table `test_hash`.** The table initially contains two partitions and the partition key is of the INT type.
 

```

-- Create the test_hash table.
gaussdb=# create table test_hash (col1 int, col2 int)
partition by hash(col1)
(
partition p1,
partition p2
);

-- Insert data.
gaussdb=# INSERT INTO test_hash VALUES(1, 1);
INSERT 0 1
gaussdb=# INSERT INTO test_hash VALUES(2, 2);
INSERT 0 1
gaussdb=# INSERT INTO test_hash VALUES(3, 3);
INSERT 0 1
gaussdb=# INSERT INTO test_hash VALUES(4, 4);
INSERT 0 1

-- View the partition information.

```

```
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_hash' AND t1.parttype = 'p';
relname | partstrategy | boundaries
```

```
-----+-----
p1 | h | {0}
p2 | h | {1}
(2 rows)
```

-- View the data.

```
gaussdb=# select * from test_hash partition (p1);
```

```
col1 | col2
-----+-----
3 | 3
4 | 4
(2 rows)
```

```
gaussdb=# select * from test_hash partition (p2);
```

```
col1 | col2
-----+-----
1 | 1
2 | 2
(2 rows)
```

-- Exchange data between the partitioned table and ordinary table.

```
gaussdb=# create table t1 (col1 int, col2 int);
```

```
CREATE TABLE
```

```
gaussdb=# alter table test_hash exchange partition (p1) with table t1;
```

```
ALTER TABLE
```

```
gaussdb=# select * from test_hash partition (p1);
```

```
col1 | col2
-----+-----
(0 rows)
```

```
gaussdb=# select * from t1;
```

```
col1 | col2
-----+-----
3 | 3
4 | 4
(2 rows)
```

-- Truncate the partition.

```
gaussdb=# alter table test_hash truncate partition p2;
```

```
ALTER TABLE
```

```
gaussdb=# select * from test_hash partition (p2);
```

```
col1 | col2
-----+-----
(0 rows)
```

-- Drop the partitioned table.

```
gaussdb=# drop table test_hash;
```

- Example 6: Create a list partitioned table **t\_multi\_keys\_list**. The table initially contains 5 partitions and the partition key is of the INT type and VARCHAR type.

-- Create the **t\_multi\_keys\_list** table.

```
gaussdb=# CREATE TABLE t_multi_keys_list (a int, b varchar(4), c int)
PARTITION BY LIST (a,b)
```

```
(
PARTITION p1 VALUES ((0,NULL)),
PARTITION p2 VALUES ((0,'1'), (0,'2'), (0,'3'), (1,'1'), (1,'2')),
PARTITION p3 VALUES ((NULL,'0'), (2,'1')),
PARTITION p4 VALUES ((3,'2'), (NULL,NULL)),
PARTITION pd VALUES (DEFAULT)
);
```

-- Create the **t\_multi\_keys\_list** table.

```
gaussdb=# DROP TABLE t_multi_keys_list;
```

## Helpful Links

[ALTER TABLE PARTITION](#) and [DROP TABLE](#)

## 7.14.79 CREATE TRIGGER

### Description

Creates a trigger. The trigger will be associated with the specified table or view, and will execute the specified function operations are performed.

### Precautions

- Currently, triggers can be created only on ordinary row-store tables, instead of on temporary tables or unlogged tables.
- If multiple triggers of the same kind are defined for the same event, they will be fired in alphabetical order by name.
- Triggers are usually used for data association and synchronization between multiple tables. SQL execution performance is greatly affected. Therefore, you are advised not to use this statement when a large amount of data needs to be synchronized and performance requirements are high.
- When a trigger meets the following conditions, the trigger statement and trigger itself can be pushed together down to a DN for execution, improving the trigger execution performance:
  - The GUC parameters **enable\_trigger\_shipping** and **enable\_fast\_query\_shipping** are enabled.
  - The trigger function used by the source table is a PL/pgSQL function (recommended).
  - The source and target tables have the same type and number of distribution keys, are both row-store tables, and belong to the same node group.
  - The **INSERT**, **UPDATE**, or **DELETE** statement on the source table contains an expression about equality comparison between all the distribution keys and the *NEW* or *OLD* variable.
  - The **INSERT**, **UPDATE**, or **DELETE** statement on the source table can be used to query shipping without a trigger.
  - There are only six types of triggers (**INSERT BEFORE FOR EACH ROW**, **INSERT AFTER FOR EACH ROW**, **UPDATE BEFORE FOR EACH ROW**, **UPDATE AFTER FOR EACH ROW**, **DELETE BEFORE FOR EACH ROW**, and **DELETE AFTER FOR EACH ROW**) on the source table, and all the triggers can be pushed down.
- The **INSERT ON DUPLICATE KEY UPDATE** statement cannot fire a trigger.
- When a trigger statement is executed, the permission is determined by the trigger creator.
- To create a trigger, you must have the **TRIGGER** permission on the specified table or have the **CREATE ANY TRIGGER** permission.
- A row-level trigger function triggered by **BEFORE** can return a **NULL** value, indicating that operations on the row are ignored. Subsequent triggers will not be executed and no **INSERT**, **UPDATE**, or **DELETE** action will be generated

on the row. The return value of the trigger function triggered by AFTER is not affected.

- For a DELETE BEFORE trigger, the return value **NEW** indicates **NULL**. For an INSERT BEFORE trigger, the return value **OLD** indicates **NULL**. For an UPDATE BEFORE trigger, the return value of the trigger function is **NULL** only when it is displayed as **NULL**.
- For a trigger function whose event is INSERT or UPDATE, the normal return value is **NEW**. If a non-NULL row is returned, the inserted or updated row is modified. For a trigger function whose event is DELETE, the normal return value is **OLD**.
- INSTEAD OF triggers can only work on views. Their trigger functions can also return **NULL**, indicating that subsequent triggers will not be executed.

## Syntax

```
CREATE [CONSTRAINT] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [OR ...] }
ON table_name
[FROM referenced_table_name]
{ NOT DEFERRABLE | [DEFERRABLE] } { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
[FOR [EACH] { ROW | STATEMENT }]
[WHEN (condition)]
EXECUTE PROCEDURE function_name (arguments);
```

Events include:

```
INSERT
UPDATE [OF column_name [, ...]]
DELETE
TRUNCATE
```

## Parameters

- **CONSTRAINT**  
(Optional) Creates a constraint trigger. That is, the trigger is used as a constraint. This is the same as a regular trigger except that the timing of the trigger firing can be adjusted using **SET CONSTRAINTS**. Constraint triggers must be AFTER ROW triggers.
- **name**  
Specifies the name of the trigger to be created. This must be distinct from the name of any other trigger for the same table. The name cannot be schema-qualified — the trigger inherits the schema of its table. For a constraint trigger, this is also the name to use when modifying the trigger's behavior using **SET CONSTRAINTS**.  
Value range: a string, which complies with the **naming convention**. A value can contain a maximum of 63 characters.
- **BEFORE**  
Specifies that the function is called before the event.
- **AFTER**  
Specifies that the function is called after the event. A constraint trigger can only be specified as **AFTER**.
- **INSTEAD OF**  
Specifies that the function is called instead of the event.

- **event**

Specifies the event that will fire the trigger. Values are **INSERT**, **UPDATE**, **DELETE**, and **TRUNCATE**. Multiple events can be specified using **OR**.

For **UPDATE** events, it is possible to specify a list of columns using this syntax:

```
UPDATE OF column_name1 [, column_name2 ...]
```

The trigger will only fire if at least one of the listed columns is mentioned as a target of the **UPDATE** statement. **INSTEAD OF UPDATE** events do not allow a list of columns.
- **table\_name**

Specifies the name of the table for which the trigger is created.

Value range: name of an existing table in the database
- **referenced\_table\_name**

Specifies the name of another table referenced by the constraint. This option is used for foreign-key constraints. It can only be specified for constraint triggers. Because foreign keys are not supported currently, this option is not recommended for general use.

Value range: name of an existing table in the database
- **DEFERRABLE | NOT DEFERRABLE**

Specifies the start time of the trigger. It can only be specified for constraint triggers. They determine whether the constraint is deferrable.

For details, see [CREATE TABLE](#).
- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

If the constraint is deferrable, the two clauses specify the default time to check the constraint. It can only be specified for constraint triggers.

For details, see [CREATE TABLE](#).
- **FOR EACH ROW | FOR EACH STATEMENT**

Specifies the frequency of firing the trigger.

  - **FOR EACH ROW** indicates that the trigger should be fired once for every row affected by the trigger event.
  - **FOR EACH STATEMENT** indicates that the trigger should be fired just once per SQL statement.

If neither is specified, the default is **FOR EACH STATEMENT**. Constraint triggers can only be marked as **FOR EACH ROW**.
- **condition**

Specifies whether the trigger function will actually be executed. If **WHEN** is specified, the function will be called only when **condition** returns **true**.

In FOR EACH ROW triggers, the WHEN condition can refer to columns of the old and/or new row values by writing **OLD.column\_name** or **NEW.column\_name** respectively. In addition, **INSERT** triggers cannot refer to **OLD**, and **DELETE** triggers cannot refer to **NEW**.

**INSTEAD OF** triggers do not support **WHEN** conditions.

Currently, **WHEN** expressions cannot contain subqueries.

Note that for constraint triggers, evaluation of the **WHEN** condition is not deferred, but occurs immediately after the row update operation is performed.

If the condition does not evaluate to **true**, then the trigger is not queued for deferred execution.

- **function\_name**

Specifies a user-defined function, which must be declared as taking no parameters and returning type trigger. This is executed when a trigger fires.

- **arguments**

Specifies an optional comma-separated list of parameters to be provided to the function when the trigger is executed. The parameters are literal string constants. Simple names and numeric constants can also be written here, but they will all be converted to strings. Check the description of the implementation language of the trigger function to find out how these parameters can be accessed within the function.

 **NOTE**

The following details trigger types:

- **INSTEAD OF** triggers must be marked as **FOR EACH ROW** and can be defined only on views.
- **BEFORE** and **AFTER** triggers on a view must be marked as **FOR EACH STATEMENT**.
- **TRUNCATE** triggers must be marked as **FOR EACH STATEMENT**.

**Table 7-164** Types of triggers supported on tables and views

| When       | Event                | Row-Level      | Statement-Level  |
|------------|----------------------|----------------|------------------|
| BEFORE     | INSERT/UPDATE/DELETE | Tables         | Tables and views |
|            | TRUNCATE             | Not supported. | Tables           |
| AFTER      | INSERT/UPDATE/DELETE | Tables         | Tables and views |
|            | TRUNCATE             | Not supported. | Tables           |
| INSTEAD OF | INSERT/UPDATE/DELETE | Views          | Not supported.   |
|            | TRUNCATE             | Not supported. | Not supported.   |

**Table 7-165** Special variables of PL/pgSQL trigger functions

| Variable | Description                                                                                                          |
|----------|----------------------------------------------------------------------------------------------------------------------|
| NEW      | New tuple for <b>INSERT</b> and <b>UPDATE</b> operations. This variable is <b>NULL</b> for <b>DELETE</b> operations. |

| Variable        | Description                                                                                                          |
|-----------------|----------------------------------------------------------------------------------------------------------------------|
| OLD             | Old tuple for <b>UPDATE</b> and <b>DELETE</b> operations. This variable is <b>NULL</b> for <b>INSERT</b> operations. |
| TG_NAME         | Trigger name.                                                                                                        |
| TG_WHEN         | Trigger timing ( <b>BEFORE</b> , <b>AFTER</b> , or <b>INSTEAD OF</b> ).                                              |
| TG_LEVEL        | Trigger frequency ( <b>ROW</b> or <b>STATEMENT</b> ).                                                                |
| TG_OP           | Trigger event ( <b>INSERT</b> , <b>UPDATE</b> , <b>DELETE</b> , or <b>TRUNCATE</b> ).                                |
| TG_RELID        | OID of the table where the trigger resides.                                                                          |
| TG_RELNAME      | Name of the table where the trigger resides. (This variable has been replaced by <b>TG_TABLE_NAME</b> .)             |
| TG_TABLE_NAME   | Name of the table where the trigger resides.                                                                         |
| TG_TABLE_SCHEMA | Schema of the table where the trigger resides.                                                                       |
| TG_NARGS        | Number of parameters for the trigger function.                                                                       |
| TG_ARGV[]       | List of parameters for the trigger function.                                                                         |

## Examples

```
-- Create a source table and a target table.
gaussdb=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

-- Create a trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
 INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
 RETURN NEW;
END
$$ LANGUAGE plpgsql;

gaussdb=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
 UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
 RETURN OLD;
END
$$ LANGUAGE plpgsql;
```

```
gaussdb=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
 DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
 RETURN OLD;
END
$$ LANGUAGE plpgsql;

-- Create an INSERT trigger.
gaussdb=# CREATE TRIGGER insert_trigger
BEFORE INSERT ON test_trigger_src_tbl
FOR EACH ROW
EXECUTE PROCEDURE tri_insert_func();

-- Create an UPDATE trigger.
gaussdb=# CREATE TRIGGER update_trigger
AFTER UPDATE ON test_trigger_src_tbl
FOR EACH ROW
EXECUTE PROCEDURE tri_update_func();

-- Create a DELETE trigger.
gaussdb=# CREATE TRIGGER delete_trigger
BEFORE DELETE ON test_trigger_src_tbl
FOR EACH ROW
EXECUTE PROCEDURE tri_delete_func();

-- Execute the INSERT event and check the trigger results.
gaussdb=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);
gaussdb=# SELECT * FROM test_trigger_src_tbl;
gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.

-- Execute the UPDATE event and check the trigger results.
gaussdb=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
gaussdb=# SELECT * FROM test_trigger_src_tbl;
gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.

-- Execute the DELETE event and check the trigger results.
gaussdb=# DELETE FROM test_trigger_src_tbl WHERE id1=100;
gaussdb=# SELECT * FROM test_trigger_src_tbl;
gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.

-- Modify a trigger.
gaussdb=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;

-- Disable insert_trigger.
gaussdb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

-- Disable all triggers on the current table.
gaussdb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;

-- Delete triggers.
gaussdb=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
gaussdb=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;
gaussdb=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

## Helpful Links

[ALTER TRIGGER](#), [DROP TRIGGER](#), and [ALTER TABLE](#)

## 7.14.80 CREATE TYPE

### Description

Defines a new data type for use in the current database. The user who defines a type becomes its owner. Types are designed only for row-store tables.

The following data types can be created: base type, shell type, enumerated type, and set type.

- **Base type**  
You can create a base type (scalar type). Generally, the functions required by a base type have to be coded in C or another low-level language.
- **Shell type**  
A shell type is simply a placeholder for a type to be defined later; it is created by issuing CREATE TYPE with no parameters except for the type name. Shell types are needed as forward references when base types are created.
- **Enumerated type**  
An enumerated type is a list of one or more quoted labels, each of which must be 1 to 64 bytes long.
- A user granted with the CREATE ANY TYPE permission can create types in the public and user schemas.

## Precautions

- If a schema name is given then the type is created in the specified schema. Otherwise, it is created in the current schema. The type name must be distinct from the name of any existing type or domain in the same schema. (Because tables have associated data types, the type name must also be distinct from the name of any existing table in the same schema.)
- When creating a non-system type by associating a function, the user needs to understand the definition of the type and the function associated with the type. If this function is not properly used, permissions may be exploited due to the associated function.

## Syntax

```
CREATE TYPE name AS
 ([attribute_name data_type [COLLATE collation] [, ...]]);

CREATE TYPE name (
 INPUT = input_function,
 OUTPUT = output_function
 [, RECEIVE = receive_function]
 [, SEND = send_function]
 [, TYPMOD_IN = type_modifier_input_function]
 [, TYPMOD_OUT = type_modifier_output_function]
 [, ANALYZE = analyze_function]
 [, INTERNALLENGTH = { internallength | VARIABLE }]
 [, PASSEDBYVALUE]
 [, ALIGNMENT = alignment]
 [, STORAGE = storage]
 [, LIKE = like_type]
 [, CATEGORY = category]
 [, PREFERRED = preferred]
 [, DEFAULT = default]
 [, ELEMENT = element]
 [, DELIMITER = delimiter]
 [, COLLATABLE = collatable]
);

CREATE TYPE name;

CREATE TYPE name AS ENUM
 (['label' [, ...]]);
```

## Parameters

### Base type

When creating a base type, you can place parameters in any order. The **input\_function** and **output\_function** parameters are required, and other parameters are optional.

- **input\_function**

Specifies the name of a function that converts data from the type's external textual form to its internal form.

The input function may be declared as taking one parameter of type `cstring` or taking three parameters of types `cstring`, `oid`, and `integer`.

- The `cstring`-type parameter is the input text as a C string.
- the second parameter is the type's own `OID`.
- The `integer`-type parameter is `typmod` of the destination column, if known (`-1` will be passed if not known).

The input function must return a value of the data type itself. Usually, an input function should be declared as **STRICT**. Otherwise, when a **NULL** input value is read and the input function is called, the first parameter is **NULL**. The function must still return **NULL** in this case, unless it raises an error. (This case is mainly meant to support domain input functions, which might need to reject **NULL** inputs.)

#### NOTE

- The input and output functions can be declared to have results or parameters of the new type, when they have to be created before the new type can be created. The type should first be defined as a shell type, which is a placeholder type that has no attributes except a name and an owner. This is done by issuing the **CREATE TYPE *name*** statement, with no additional parameters. Then the I/O functions can be defined referencing the shell type. Finally, **CREATE TYPE** with a full definition replaces the shell entry with a complete, valid type definition, after which the new type can be used normally.
- If the input and output functions are internal functions and are specified as internal system functions, the parameter types of the input and output functions must be the same, and the parameter types of **INTERNALLENGTH** and **PASSEDBYVALUE** of the new type must be the same as those of the input and output functions.
- **output\_function**  
Specifies the name of a function that converts data from the type's internal form to its external textual form.  
The output function must be declared as taking one parameter of the new data type. The output function must return type `cstring`. Output functions are not called for **NULL** values.
- **receive\_function**  
(Optional) Specifies the name of a function that converts data from the type's external binary form to its internal form.  
If this function is not supplied, the type cannot participate in binary input. The binary representation should be chosen to be cheap to convert to internal form, while being reasonably portable. (For example, the standard integer data types use network byte order as the external binary representation, while

the internal representation is in the machine's native byte order.) The receive function should perform adequate checking to ensure that the value is valid.

The receive function may be declared as taking one parameter of type `internal` or taking three parameters of types `internal`, `oid`, `integer`.

- The `internal`-type parameter is a pointer to a `StringInfo` buffer holding the received byte strings.
- The `oid`- and `integer`-type parameters are the same as those of the text input function.

The receive function must return a value of the data type itself. Usually, a receive function should be declared **STRICT**; if it is not, it will be called with a **NULL** first parameter when reading a **NULL** input value. The function must still return **NULL** in this case, unless it raises an error. (This case is mainly meant to support domain receive functions, which might need to reject **NULL** inputs.)

- **send\_function**

(Optional) Specifies the name of a function that converts data from the type's internal form to its external binary form.

If this function is not supplied, the type cannot participate in binary output. The send function must be declared as taking one parameter of the new data type. The send function must return `bytea`. Send functions are not called for **NULL** values.

- **type\_modifier\_input\_function**

(Optional) Specifies the name of a function that converts an array of modifiers for a type to its internal format.

- **type\_modifier\_output\_function**

(Optional) Specifies the name of a function that converts the internal format of modifiers for a type to its external text format.

 **NOTE**

**type\_modifier\_input\_function** and **type\_modifier\_output\_function** are needed if the type supports modifiers, that is optional constraints attached to a type declaration, such as `char(5)` or `numeric(30,2)`. GaussDB allows user-defined types to take one or more simple constants or identifiers as modifiers. However, this information must be capable of being packed into a single non-negative integer value for storage in the system catalogs. Declared modifiers are passed to **type\_modifier\_input\_function** in the `cstring` array format. It must check the values for validity (throwing an error if they are wrong), and if they are correct, return a single non-negative integer value that will be stored as the column `"typmod"`. Type modifiers will be rejected if the type does not have a **type\_modifier\_input\_function**. The **type\_modifier\_output\_function** converts the internal integer `typmod` value back to the correct form for user display. It must return a `cstring` value that is the exact string to append to the type name; for example `numeric`'s function might return `(30,2)`. It is allowed to omit the **type\_modifier\_output\_function**, in which case the default display format is just the stored `typmod` integer value enclosed in parentheses.

- **analyze\_function**

(Optional) Specifies the name of a function that performs statistical analysis for the data type.

By default, **ANALYZE** will attempt to gather statistics using the type's "equals" and "less-than" operators, if there is a default B-tree operator class for the type. For non-scalar types, this behavior is likely to be unsuitable, so it can be overridden by specifying a custom analysis function. The analysis function

must be declared to take one parameter of type `internal` and return a boolean result.

- **internallength**

(Optional) Specifies the length in bytes of the new type's internal representation. The default assumption is that it is variable-length.

While the details of the new type's internal representation are only known to the I/O functions and other functions you create to work with the type, there are several attributes of the internal representation that must be declared to GaussDB. Foremost of these is **internallength**. Base data types can be fixed-length, in which case **internallength** is a positive integer, or variable length, indicated by setting **internallength** to **VARIABLE**. (Internally, this is represented by setting **typlen** to **-1**.) The internal representation of all variable-length types must start with a 4-byte integer giving the total length of this value of the type.

- **PASSEDBYVALUE**

(Optional) Indicates that values of this data type are passed by value, rather than by reference. You cannot pass by value types whose internal representation is larger than the size of the Datum type (4 bytes on most machines, 8 bytes on a few).

- **alignment**

(Optional) Specifies the storage alignment requirement of the data type. If specified, it must be **char**, **int2**, **int4**, or **double**; the default is **int4**.

The allowed values equate to alignment on 1, 2, 4, or 8 byte boundaries. Note that variable-length types must have an alignment of at least 4, since they necessarily contain an `int4` as their first component.

- **storage**

(Optional) Specifies the storage strategy for the data type.

If specified, it must be **plain**, **external**, **extended**, or **main**; the default is **plain**.

- **plain** specifies that data of the type will always be stored in-line and not compressed. (Only **plain** is allowed for fixed-length types.)
- **extended** specifies that the system will first try to compress a long data value, and will move the value out of the main table row if it is still too long.
- **external** allows the value to be moved out of the main table, but the system will not try to compress it.
- **main** allows compression, but discourages moving the value out of the main table. (Data items with this storage strategy might still be moved out of the main table if there is no other way to make a row fit, but they will be kept in the main table preferentially over **extended** and **external** items.)

All **storage** values other than **plain** imply that the functions of the data type can handle values that have been toasted. The specific other value given merely determines the default **TOAST** storage strategy for columns of a toastable data type; users can pick other strategies for individual columns using **ALTER TABLE SET STORAGE**.

- **like\_type**

(Optional) Specifies the name of an existing data type that the new type will have the same representation as. The values of **internallength**, **passedbyvalue**, **alignment**, and **storage** are copied from that type, unless overridden by explicit specification elsewhere in this **CREATE TYPE** statement.

Specifying representation in this way is especially useful when the underlying implementation of a new type references an existing type.

- **category**

(Optional) Specifies the category code (a single ASCII character) for this type. The default is **U** for a user-defined type. You may also choose other ASCII characters to create custom categories.

- **preferred**

(Optional) Specifies whether a type is preferred within its type category. If it is, the value will be **TRUE**, else **FALSE**. The default is **FALSE**. Be very careful about creating a preferred type within an existing type category, as this could cause surprising changes in behavior.

 **NOTE**

The **category** and **preferred** parameters can be used to help control which implicit cast will be applied in ambiguous situations. Each data type belongs to a category named by a single ASCII character, and each type is either preferred or not within its category. The parser will prefer casting to preferred types (but only from other types within the same category) when this rule is helpful in resolving overloaded functions or operators. For types that have no implicit casts to or from any other types, it is sufficient to leave these settings at the defaults. However, for a group of related types that have implicit casts, it is often helpful to mark them all as belonging to a category and select one or two of the most general types as being preferred within the category. The **category** parameter is especially useful when adding a user-defined type to an existing built-in category, such as the numeric or string types. However, it is also possible to create entirely-user-defined type categories. Select any ASCII character other than an uppercase letter to name such a category.

- **default**

(Optional) Specifies the default value for the data type. If this is omitted, the default is null.

A default value can be specified, in case a user wants columns of the data type to default to something other than the null value. Specify the default with the **DEFAULT** keyword. (Such a default can be overridden by an explicit **DEFAULT** clause attached to a particular column.)

- **element**

(Optional) Specifies the type of array elements when an array type is created. For example, to define an array of 4-byte integers (int4), specify **ELEMENT = int4**.

- **delimiter**

(Optional) Specifies the delimiter character to be used between values in arrays made of this type.

**delimiter** can be set to a specific character. The default delimiter is the comma (,).

- **collatable**

(Optional) Specifies whether this type's operations can use collation information. If they can, the value will be **TRUE**, else **FALSE** (default).

If **collatable** is **TRUE**, column definitions and expressions of the type may carry collation information through use of the **COLLATE** clause. It is up to the implementations of the functions operating on the type to actually make use of the collation information; this does not happen automatically merely by marking the type collatable.

- **label**

(Optional) Represents the textual label associated with one value of an enumerated type. It is a string of 1 to 63 characters.

## Examples

```
gaussdb=# CREATE TABLE t1_compfoo(a int, b compfoo);
gaussdb=# CREATE TABLE t2_compfoo(a int, b compfoo);

-- Change the owner of the user-defined type compfoo1 to usr1.
gaussdb=# CREATE USER usr1 PASSWORD '*****';

Delete related tables and users.
gaussdb=# DROP TABLE t1_compfoo;
gaussdb=# DROP TABLE t2_compfoo;
gaussdb=# DROP SCHEMA usr1;
gaussdb=# DROP USER usr1;

-- Create an enumerated type.
gaussdb=# CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');

-- Add a label.
gaussdb=# ALTER TYPE bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';

-- Rename a label.
gaussdb=# ALTER TYPE bugstatus RENAME VALUE 'create' TO 'new';

-- Compile the .so file and create a shell type.
gaussdb=# CREATE TYPE complex;
-- This statement creates a placeholder for the type to be defined so that the type can be referenced when
its I/O functions are defined. Then, you can define I/O functions. Note that the functions must be declared
to take the NOT FENCED mode during creation.
gaussdb=# CREATE FUNCTION
complex_in(cstring)
 RETURNS complex
 AS 'filename'
 LANGUAGE C IMMUTABLE STRICT not fenced;

gaussdb=# CREATE FUNCTION
complex_out(complex)
 RETURNS cstring
 AS 'filename'
 LANGUAGE C IMMUTABLE STRICT not fenced;

gaussdb=# CREATE FUNCTION
complex_rcv(internal)

RETURNS complex

AS 'filename'

LANGUAGE C IMMUTABLE STRICT not fenced;

gaussdb=# CREATE FUNCTION
complex_send(complex)

RETURNS bytea

AS 'filename'
```

```
LANGUAGE C IMMUTABLE STRICT not fenced;
-- Finally, provide a complete definition of the data type.
gaussdb=# CREATE TYPE complex (

internallength = 16,

input = complex_in,

output = complex_out,

receive = complex_rcv,

send = complex_send,

alignment = double
);
```

The C functions corresponding to the input, output, receive, and send functions are defined as follows:

```
-- Define a structure body Complex.
typedef struct Complex {
 double x;
 double y;
} Complex;

-- Define an input function.
PG_FUNCTION_INFO_V1(complex_in);

Datum
complex_in(PG_FUNCTION_ARGS)
{
 char *str = PG_GETARG_CSTRING(0);
 double x,
 y;
 Complex *result;

 if (sscanf(str, " (%lf , %lf)", &x, &y) != 2)
 ereport(ERROR,
 (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
 errmsg("invalid input syntax for complex: \"%s\"",
 str)));

 result = (Complex *) palloc(sizeof(Complex));
 result->x = x;
 result->y = y;
 PG_RETURN_POINTER(result);
}

-- Define an output function.
PG_FUNCTION_INFO_V1(complex_out);

Datum
complex_out(PG_FUNCTION_ARGS)
{
 Complex *complex = (Complex *) PG_GETARG_POINTER(0);
 char *result;

 result = (char *) palloc(100);
 snprintf(result, 100, "(%g,%g)", complex->x, complex->y);
 PG_RETURN_CSTRING(result);
}

-- Define a receive function.
PG_FUNCTION_INFO_V1(complex_rcv);

Datum
complex_rcv(PG_FUNCTION_ARGS)
{
```

```
StringInfo buf = (StringInfo) PG_GETARG_POINTER(0);
Complex *result;

result = (Complex *) palloc(sizeof(Complex));
result->x = pq_getmsgfloat8(buf);
result->y = pq_getmsgfloat8(buf);
PG_RETURN_POINTER(result);
}

-- Define a send function.
PG_FUNCTION_INFO_V1(complex_send);

Datum
complex_send(PG_FUNCTION_ARGS)
{
 Complex *complex = (Complex *) PG_GETARG_POINTER(0);
 StringInfoData buf;

 pq_begintypsend(&buf);
 pq_sendfloat8(&buf, complex->x);
 pq_sendfloat8(&buf, complex->y);
 PG_RETURN_BYTEA_P(pq_endtypsend(&buf));
}
```

## Helpful Links

[ALTER TYPE](#) and [DROP TYPE](#)

## 7.14.81 CREATE USER

### Description

Creates a user.

### Precautions

- A user created using the CREATE USER statement has the LOGIN permission by default.
- When you run the **CREATE USER** command to create a user, the system creates a schema with the same name as the user in the database where the command is executed.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.

### Syntax

```
CREATE USER user_name [[WITH] option [...]] [ENCRYPTED | UNENCRYPTED] { PASSWORD | IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```

The option clause is used to configure information, including permissions and properties.

```
{SYSADMIN | NOSYSADMIN}
| {MONADMIN | NOMONADMIN}
| {OPRADMIN | NOOPRADMIN}
| {POLADMIN | NOPOLADMIN}
| {AUDITADMIN | NOAUDITADMIN}
| {CREATEDB | NOCREATEDB}
| {USEFT | NOUSEFT}
| {CREATEROLE | NOCREATEROLE}
```

```
| {INHERIT | NOINHERIT}
| {LOGIN | NOLOGIN}
| {REPLICATION | NOREPLICATION}
| {VCADMIN | NOVCADMIN}
| {PERSISTENCE | NOPERSISTENCE}
| CONNECTION LIMIT connlimit
| VALID BEGIN 'timestamp'
| VALID UNTIL 'timestamp'

| USER GROUP 'groupuser'
| PERM SPACE 'spacelimit'
| TEMP SPACE 'tmpspacelimit'
| SPILL SPACE 'spillspacelimit'
| NODE GROUP logic_cluster_name
| IN ROLE role_name [, ...]
| IN GROUP role_name [, ...]
| ROLE role_name [, ...]
| ADMIN role_name [, ...]
| USER role_name [, ...]
| SYSID uid
| DEFAULT TABLESPACE tablespace_name
| PROFILE DEFAULT
| PROFILE profile_name
| PGUSER
```

## Parameters

- **user\_name**

Username.

Value range: a string that complies with the [Identifier Naming Conventions](#). A value can contain a maximum of 63 characters. If a username contains uppercase letters, the database automatically converts the uppercase letters into lowercase letters. To create a username that contains uppercase letters, enclose the username with double quotation marks ("").

- **password**

Specifies the login password.

The new password must:

- Contain at least eight characters. This is the default length.
- Differ from the username or the username spelled backward.
- Contain at least three of the following character types: uppercase characters, lowercase characters, digits, and special characters (limited to ~!@#%&^&\*( )-\_=+\|[{}];:;<.>/?). If the password contains characters other than the preceding characters, an error will be reported during statement execution.
- The password can also be a ciphertext character string that meets the format requirements. This mode is mainly used to import user data. You are advised not to use it directly. If a ciphertext password is used, you need to know the plaintext corresponding to the ciphertext password and ensure a complex plaintext password. The database does not verify the complexity of the ciphertext password, so you should ensure the password security.
- When creating a user, enclose the user password in single quotation marks.

Value range: a string.

For details about other parameters of CREATE USER, see [Parameters](#) in "CREATE ROLE."

## Examples

```
-- Create user jim whose login password is *****.
gaussdb=# CREATE USER jim PASSWORD '*****';

-- Alternatively, you can run the following statement:
gaussdb=# CREATE USER kim IDENTIFIED BY '*****';

-- To create a user with the CREATEDB permission, add the CREATEDB keyword.
gaussdb=# CREATE USER dim CREATEDB PASSWORD '*****';

-- Change the login password of user jim.
gaussdb=# ALTER USER jim IDENTIFIED BY '*****' REPLACE '*****';

-- Add the CREATEROLE permission to jim.
gaussdb=# ALTER USER jim CREATEROLE;

-- Lock jim.
gaussdb=# ALTER USER jim ACCOUNT LOCK;

-- Drop users.
gaussdb=# DROP USER kim CASCADE;
gaussdb=# DROP USER jim CASCADE;
gaussdb=# DROP USER dim CASCADE;
```

## Helpful Links

[ALTER USER](#), [CREATE ROLE](#), and [DROP USER](#)

## 7.14.82 CREATE VIEW

### Description

Creates a view. A view is a virtual table, not a base table. Only view definition is stored in the database and view data is not. The data is stored in a base table. If data in the base table changes, the data in the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database.

### Precautions

A user granted the CREATE ANY TABLE permission can create views in the public and user schemas.

### Syntax

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW view_name [(column_name [, ...])]
[WITH ({view_option_name [= view_option_value]} [, ...])]
AS query;
```

#### NOTE

You can use **WITH(security\_barrier)** to create a relatively secure view. This prevents attackers from printing base table data by using the **RAISE** statement of low-cost functions.

After a view is created, you are not allowed to use REPLACE to modify column names in the view or delete the columns.

### Parameters

- **OR REPLACE**

- Redefines the view if it already exists.
- **TEMP | TEMPORARY**  
Creates a temporary view.
  - **view\_name**  
Specifies the name (optionally schema-qualified) of the view to be created.  
Value range: a string. It must comply with the [naming convention](#).
  - **column\_name**  
Specifies an optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.  
Value range: a string. It must comply with the [naming convention](#).
  - **view\_option\_name [= view\_option\_value]**  
Specifies an optional parameter for a view.  
Currently, **view\_option\_name** supports only the **security\_barrier** parameter. This parameter is used when the view attempts to provide row-level security.  
Value range: Boolean type. It can be **TRUE** or **FALSE**.
  - **query**  
Specifies a **SELECT** or **VALUES** statement that will provide the columns and rows of the view.

---

**NOTICE**

If **query** contains a clause specifying the partition of a partitioned table, the OID of the specified partition is hardcoded to the system catalog when the view is created. If the partition DDL syntax that causes the change in the OID of the specified partition is used, for example, DROP, SPLIT, or MERGE, the view is unavailable. In this case, you need to create a view.

---

## Examples

```
-- Create a view consisting of columns whose spcname is pg_default.
gaussdb=# CREATE VIEW myView AS
 SELECT * FROM pg_tablespace WHERE spcname = 'pg_default';

-- Query the view.
gaussdb=# SELECT * FROM myView ;

-- Delete the view.
gaussdb=# DROP VIEW myView;
```

## Helpful Links

[ALTER VIEW](#) and [DROP VIEW](#)

## 7.14.83 CREATE WEAK PASSWORD DICTIONARY

### Function

**CREATE WEAK PASSWORD DICTIONARY** inserts one or more weak passwords into the **gs\_global\_config** table.

## Precautions

- Only the initial user, system administrator, and security administrator have the permission to execute this syntax.
- Passwords in the weak password dictionary are stored in the **gs\_global\_config** system catalog.
- The weak password dictionary is empty by default. You can use this syntax to add one or more weak passwords.
- When a user attempts to execute this syntax to insert a weak password that already exists in the **gs\_global\_config** table, only one weak password is retained in the table.

## Syntax

```
CREATE WEAK PASSWORD DICTIONARY
[WITH VALUES] ({'weak_password'} [, ...]);
```

## Parameter Description

weak\_password

Weak password

Value range: a character string.

## Example

```
-- Insert a single weak password into the gs_global_config system catalog.
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1');

-- Insert multiple weak passwords into the gs_global_config system catalog.
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password2'),('password3');

-- Clear all weak passwords in the gs_global_config system catalog.
gaussdb=# DROP WEAK PASSWORD DICTIONARY;

-- View existing weak passwords.
gaussdb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
```

## Helpful Links

[13.14.119-DROP WEAK PASSWORD DICTIONARY](#)

## 7.14.84 CURSOR

### Description

This statement is used to create a cursor and retrieve specified rows of data from a query.

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

## Precautions

- **CURSOR** is used only in transaction blocks.
- Generally, **CURSOR** and **SELECT** both have text returns. Since data is stored in binary format in the system, the system needs to convert the data from the binary format to the text format. If data is returned in text format, client applications need to convert the data back to the binary format for processing. **FETCH** implements conversion between binary data and text data.
- Binary cursors should be used carefully. Text usually occupies larger space than binary data. A binary cursor returns internal binary data, which is easier to operate. A text cursor returns text, which is easier to retrieve and therefore reduces workload on the client. As an example, if a query returns a value of one from an integer column, you would get a string of 1 with a default cursor, whereas with a binary cursor you would get a 4-byte field containing the internal representation of the value (in big-endian byte order).

## Syntax

```
CURSOR cursor_name
[BINARY] [NO SCROLL] [{ WITH | WITHOUT } HOLD]
FOR query ;
```

## Parameters

- **cursor\_name**  
Specifies the name of the cursor to be created.  
Value range: a string. It must comply with the naming convention.
- **BINARY**  
Causes the cursor to return data in binary rather than in text format.
- **NO SCROLL**  
Specifies how the cursor retrieves rows.
  - **NO SCROLL**: specifies that the cursor cannot be used to retrieve rows in a nonsequential fashion.
  - Unspecified: Based on the query's execution plan, the system automatically determines whether the cursor can be used to retrieve rows in a nonsequential fashion.
- **WITH HOLD | WITHOUT HOLD**  
Specifies whether the cursor can continue to be used after the transaction that created it successfully commits.
  - **WITH HOLD**: The cursor can continue to be used after the transaction that created it successfully commits.
  - **WITHOUT HOLD**: The cursor cannot be used outside of the transaction that created it.
  - If neither **WITH HOLD** nor **WITHOUT HOLD** is specified, the default is **WITHOUT HOLD**.
  - Cross-node transactions (for example, DDL-contained transactions created in a coordinator cluster) do not support **WITH HOLD**.
- **query**  
Uses a **SELECT** or **VALUES** clause to specify the rows to be returned by the cursor.

Value range: **SELECT** or **VALUES** clause

## Examples

See [Examples](#) in section "FETCH."

## Helpful Links

[FETCH](#)

## 7.14.85 DEALLOCATE

### Description

DEALLOCATE deallocates prepared statements.

### Precautions

- If you do not explicitly deallocate a prepared statement, it is deallocated when the session ends.
- The PREPARE keyword in the syntax is always ignored.

### Syntax

```
DEALLOCATE [PREPARE] { name | ALL };
```

### Parameters

- **name**  
Specifies the name of the prepared statement to be deallocated.
- **ALL**  
Deallocates all prepared statements.

### Examples

None

## 7.14.86 DECLARE

### Description

**DECLARE** defines a cursor to retrieve a small number of rows at a time out of a larger query and can be the start of an anonymous block.

This section describes usage of cursors. The usage of anonymous blocks is available in [BEGIN](#).

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

Generally, CURSOR and SELECT both have text returns. Since data is stored in binary format in the system, the system needs to convert the data from the binary

format to the text format. If data is returned in text format, client applications need to convert the data back to the binary format for processing. **FETCH** implements conversion between binary data and text data.

## Precautions

- **CURSOR** is used only in transaction blocks.
- Binary cursors should be used carefully. Text usually occupies larger space than binary data. A binary cursor returns internal binary data, which is easier to operate. A text cursor returns text, which is easier to retrieve and therefore reduces workload on the client. As an example, if a query returns a value of one from an integer column, you would get a string of 1 with a default cursor, whereas with a binary cursor you would get a 4-byte field containing the internal representation of the value (in big-endian byte order).

## Syntax

- Define a cursor.

```
DECLARE cursor_name [BINARY] [NO SCROLL]
 CURSOR [{ WITH | WITHOUT } HOLD] FOR query ;
```
- Enable an anonymous block.

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

## Parameters

- **cursor\_name**  
Specifies the name of the cursor to be created.  
Value range: a string. It must comply with the naming convention.
- **BINARY**  
Causes the cursor to return data in binary rather than in text format.
- **NO SCROLL**  
Specifies how the cursor retrieves rows.
  - **NO SCROLL**: specifies that the cursor cannot be used to retrieve rows in a nonsequential fashion.
  - Unspecified: Based on the query's execution plan, the system automatically determines whether the cursor can be used to retrieve rows in a nonsequential fashion.
- **WITH HOLD**  
**WITHOUT HOLD**  
Specifies whether the cursor can continue to be used after the transaction that created it successfully commits.
  - **WITH HOLD**: The cursor can continue to be used after the transaction that created it successfully commits.
  - **WITHOUT HOLD**: The cursor cannot be used outside of the transaction that created it.
  - If neither **WITH HOLD** nor **WITHOUT HOLD** is specified, the default is **WITHOUT HOLD**.

#### NOTICE

- For a cursor declared as **WITH HOLD**, all data of the cursor is cached when a transaction ends. If the cursor has a large amount of data, this process may take a long time.

- **query**  
Uses a **SELECT** or **VALUES** clause to specify the rows to be returned by the cursor.  
Value range: **SELECT** or **VALUES** clause
- **declare\_statements**  
Declares a variable, including its name and type, for example, **sales\_cnt int**.
- **execution\_statements**  
Specifies the statement to be executed in an anonymous block.  
Value range: an existing function name

## Examples

For details about how to start an anonymous block, see [Examples](#) in section "BEGIN."

For details about how to define a cursor, see [Examples](#) in section "FETCH."

## Helpful Links

[BEGIN](#) and [FETCH](#)

## 7.14.87 DELETE

### Description

Deletes rows that satisfy the WHERE clause from the specified table. If the WHERE clause is absent, the effect is to delete all rows in the table. The result is a valid, but an empty table.

### Precautions

- The owner of a table, users granted with the DELETE permission on the table, or users granted with the DELETE ANY TABLE permission can delete data from the table. System administrators have the permission to delete data from the table by default, as well as the SELECT permission on any table in the USING clause or whose values are read in **condition**.
- For row-store replication tables, DELETE can be performed only in the following two scenarios:
  - Scenarios with primary key constraints.
  - Scenarios where the execution plan can be pushed down.
- For the DELETE statement whose subquery is a stream plan, the same deleted row cannot be concurrently updated.

## Syntax

```
[WITH [RECURSIVE] with_query [, ...]]
DELETE [/*+ plan_hint */] [FROM] [ONLY] table_name [*] [[AS] alias]
 [USING using_list]
 [WHERE condition | WHERE CURRENT OF cursor_name]
 [ORDER BY {expression [ASC | DESC | USING operator]}]
 [LIMIT { count }]
 [RETURNING { * | { output_expr [[AS] output_name] } [, ...] }];
```

Format of **with\_query**:

```
with_query_name [(column_name [, ...])] AS [[NOT] MATERIALIZED]
({select | values | insert | update | delete})
```

## Parameters

- **WITH [ RECURSIVE ] with\_query [, ...]**

Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table.

If RECURSIVE is specified, it allows a SELECT subquery to reference itself by name.

  - **with\_query\_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.
  - **column\_name** specifies the column name displayed in the subquery result set.
  - Each subquery can be a **SELECT**, **VALUES**, **INSERT**, **UPDATE** or **DELETE** statement.
  - You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE.
    - If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.
    - If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the trunk statement to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.
- **plan\_hint**

Follows the **DELETE** keyword in the */\*+ \*/* format. It is used to optimize the plan of a **DELETE** statement block. For details, see [Hint-based Tuning](#). In each statement, only the first */\*+ plan\_hint \*/* comment block takes effect as a hint. Multiple hints can be written.
- **ONLY**

If **ONLY** is specified before the table name, matching rows are deleted from the named table only. If **ONLY** is not specified, matching rows are also deleted from any tables inheriting from the named table.

- **table\_name**  
Specifies the name (optionally schema-qualified) of the table to delete rows from.  
Value range: an existing table name  
 **NOTE**  
You can use database links to perform operations on remote tables. For details, see [DATABASE LINK](#).
- **alias**  
Specifies a substitute name for the target table.  
Value range: a string. It must comply with the [naming convention](#).
- **using\_list**  
Specifies the **USING** clause.
- **condition**  
Specifies an expression that returns a value of type Boolean. Only rows for which this expression returns **true** will be deleted. You are advised not to use numeric types such as int as conditions, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and 0 is implicitly converted to **false**), which may cause unexpected results.
- **WHERE CURRENT OF cursor\_name**  
When the cursor points to a row in a table, you can use this syntax to delete the row. For details about the restrictions, see [UPDATE](#).
- **ORDER BY**  
For details about the keywords, see [SELECT](#).
- **LIMIT**  
For details about the keywords, see [SELECT](#).
- **output\_expr**  
Specifies an expression to be computed and returned by the **DELETE** statement after each row is deleted. The expression can use any column names of the table. You can use \* to return all columns.
- **output\_name**  
Specifies a name to use for a returned column.  
Value range: a string. It must comply with the [naming convention](#).

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer_address table.
gaussdb=# CREATE TABLE tpcds.customer_address
(
ca_address_sk INTEGER NOT NULL,
ca_address_id CHARACTER(16) NOT NULL,
ca_street_number INTEGER
ca_street_name CHARACTER (20)
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.customer_address VALUES (1, 'AAAAAAAABAAAAAAA', '18', 'Jackson'),
(10000, 'AAAAAAAACAAAAAAA', '362', 'Washington 6th'),(15000, 'AAAAAAAADAAAAAAA', '585', 'Dogwood
```

```
Washington');

-- Create the tpcds.customer_address_bak table.
gaussdb=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;

-- Delete employees whose ca_address_sk is smaller than 14888 from the tpcds.customer_address_bak
table.
gaussdb=# DELETE FROM tpcds.customer_address_bak WHERE ca_address_sk < 14888;

-- Delete all data from the tpcds.customer_address_bak table.
gaussdb=# DELETE FROM tpcds.customer_address_bak;

-- Delete the tpcds.customer_address_bak table.
gaussdb=# DROP TABLE tpcds.customer_address_bak;

-- Delete the tpcds.customer_address table.
gaussdb=# DROP TABLE tpcds.customer_address;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Suggestions

- delete  
To delete all records in a table, use the TRUNCATE syntax.

## 7.14.88 DO

### Description

Executes an anonymous code block.

A code block is regarded as a function body without parameters. The return value type is void. It is parsed and executed a single time.

### Precautions

- The procedural language to be used must already have been installed into the current database by means of **CREATE LANGUAGE**. **plpgsql** is installed by default, but other languages are not.
- To use an untrusted language, you must have the USAGE permission on the programming language or the SYSADMIN permission.

### Syntax

```
DO [LANGUAGE lang_name] code;
```

### Parameters

- **lang\_name**  
Specifies the name of the procedural language the code is written in. If omitted, the default is **plpgsql**.
- **code**  
Specifies the programming language code that can be executed. The value must be a character string.

## Examples

```
-- Create the webuser user.
gaussdb=# CREATE USER webuser PASSWORD '*****';

-- Grant all permissions on all views in the tpcds schema to the webuser user.
gaussdb=# DO $$DECLARE r record;
BEGIN
 FOR r IN SELECT c.relname,n.nspname FROM pg_class c,pg_namespace n
 WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
 LOOP
 EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO
webuser';
 END LOOP;
END$$;

-- Delete the webuser user.
gaussdb=# DROP USER webuser CASCADE;
```

## 7.14.89 DROP AUDIT POLICY

### Description

Deletes an audit policy.

### Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

### Syntax

```
DROP AUDIT POLICY [IF EXISTS] policy_name;
```

### Parameters

- **policy\_name**  
Specifies the audit policy name, which must be unique.  
Value range: a string. It must comply with the [naming convention](#).

### Examples

See [Examples](#) in section "CREATE AUDIT POLICY."

### Helpful Links

[CREATE AUDIT POLICY](#) and [ALTER AUDIT POLICY](#)

## 7.14.90 DROP CLIENT MASTER KEY

### Description

Drops a CMK.

## Precautions

- Only the CMK owner or a user who has been granted the DROP permission can run this command. System administrators have this permission by default.
- This command can only be used to delete the metadata information recorded in the system catalog of the database, but cannot be used to delete the CMK file. You need to use KeyTool to delete the CMK file.

## Syntax

```
DROP CLIENT MASTER KEY [IF EXISTS] client_master_key_name [, ...] [CASCADE | RESTRICT];
```

## Parameters

- **IF EXISTS**  
If a specified CMK does not exist, a notice rather than an error is issued.
- **client\_master\_key\_name**  
Name of a CMK to be deleted.  
Value range: a string. It is the name of an existing CMK.
- **CASCADE | RESTRICT**  
Allows/Restricts to delete objects that depend on the CMK.

## Examples

```
-- Delete a CMK object.
gaussdb=> DROP CLIENT MASTER KEY imgCMK CASCADE;
NOTICE: drop cascades to column setting: imgcek
DROP GLOBAL SETTING
```

## 7.14.91 DROP COLUMN ENCRYPTION KEY

### Description

Drops a CEK.

### Precautions

Only the CEK owner or a user who has been granted the DROP permission can run this command. System administrators have this permission by default.

### Syntax

```
DROP COLUMN ENCRYPTION KEY [IF EXISTS] client_column_key_name [, ...] [CASCADE | RESTRICT];
```

### Parameters

- **IF EXISTS**  
If a specified CEK does not exist, a notice rather than an error is issued.
- **client\_column\_key\_name**  
Name of a CEK to be deleted.  
Value range: a string. It is the name of an existing CEK.
- **CASCADE | RESTRICT**

For fully-encrypted databases, this syntax is high-risk operation. Actually, encrypted columns that depend on CEKs cannot be deleted.

## Examples

```
-- Delete a CEK object.
gaussdb=# DROP COLUMN ENCRYPTION KEY imgCEK CASCADE;
ERROR: cannot drop column setting: imgcek cascadelly because encrypted column depend on it.
HINT: we have to drop encrypted column: name, ... before drop column setting: imgcek cascadelly.
```

## 7.14.92 DROP DATABASE

### Description

Deletes a database.

### Precautions

- Only the database owner or a user granted with the DROP permission can run the **DROP DATABASE** command. System administrators have this permission by default.
- The preinstalled POSTGRES, TEMPLATE0, and TEMPLATE1 databases are protected and therefore cannot be deleted. To check databases in the current service, run the gsql statement `\l`.
- If any users are connected to the database, the database cannot be deleted. To check the current database connections, open the DV\_SESSIONS view.
- DROP DATABASE cannot be executed within a transaction block.
- Before deleting a database, run the **CLEAN CONNECTION TO ALL FORCE FOR DATABASE XXXX** command to forcibly stop the existing user connections and backend threads, preventing database deletion failures caused by running backend threads. Forcibly stopping backend threads may cause data inconsistency in the current database. Therefore, execute this command only when you are sure to delete the database.
- If DROP DATABASE fails and is rolled back, run **DROP DATABASE IF EXISTS** again.

---

#### NOTICE

DROP DATABASE cannot be undone.

---

### Syntax

```
DROP DATABASE [IF EXISTS] database_name ;
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified database does not exist.
- **database\_name**  
Specifies the name of the database to be deleted.  
Value range: an existing database name

## Examples

See [Examples](#) in section "CREATE DATABASE."

## Helpful Links

[CREATE DATABASE](#)

## Suggestions

- DROP DATABASE  
Do not delete databases during transactions.

## 7.14.93 DROP DATABASE LINK

### Description

Drops database link objects.

### Syntax

```
DROP [PUBLIC] DATABASE LINK dblink ;
```

### Parameters

- dblink  
Name of a connection object.
- PUBLIC  
Connection type. If **PUBLIC** is not specified, the database link is private by default.

## 7.14.94 DROP DIRECTORY

### Description

Deletes a directory.

### Precautions

- When **enable\_access\_server\_directory** is set to **off**, only the initial user is allowed to delete the directory object.
- When **enable\_access\_server\_directory** is set to **on**, a user with the SYSADMIN permission, a directory object owner, a user granted with the DROP permission for a directory, or a user who inherits permissions from the built-in role gs\_role\_directory\_drop can delete a directory object.

### Syntax

```
DROP DIRECTORY [IF EXISTS] directory_name;
```

### Parameters

- **directory\_name**

Specifies the name of the directory to be deleted.  
Value range: an existing directory name

## Examples

```
-- Create a directory.
gaussdb=# CREATE OR REPLACE DIRECTORY dir as '/tmp/';

-- Delete a foreign table.
gaussdb=# DROP DIRECTORY dir;
```

## Helpful Links

[CREATE DIRECTORY](#) and [ALTER DIRECTORY](#)

## 7.14.95 DROP FOREIGN DATA WRAPPER

### Description

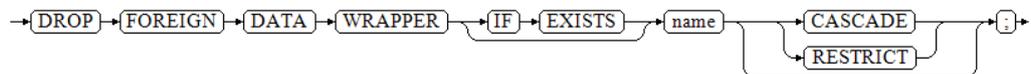
Deletes a specified foreign data wrapper.

### Precautions

The DROP statement can be successfully executed only when **support\_extended\_features** is set to **on**.

### Syntax

```
DROP FOREIGN DATA WRAPPER [IF EXISTS] name [CASCADE | RESTRICT];
```



### Parameters

- **IF EXISTS**  
Sends a notification instead of throwing an error if the foreign data wrapper does not exist.
- **name**  
Specifies the name of an existing foreign data wrapper.
- **CASCADE**  
Automatically drops objects (such as servers) that depend on the foreign data wrapper.
- **RESTRICT**  
If there are objects that depend on the foreign data wrapper, the foreign data wrapper cannot be deleted. This is the default behavior.

### Example

```
-- Delete the foreign data wrapper dbi.
gaussdb=# DROP FOREIGN DATA WRAPPER dbi;
```

## Helpful Links

[ALTER FOREIGN DATA WRAPPER](#) and [CREATE FOREIGN DATA WRAPPER](#)

## 7.14.96 DROP FUNCTION

### Description

Deletes a function.

### Precautions

If a function involves operations on temporary tables, DROP FUNCTION cannot be used.

Only the function owner or a user granted with the DROP permission can run the **DROP FUNCTION** command. By default, system administrators have the permission.

### Syntax

```
DROP FUNCTION [IF EXISTS] function_name
[(([argname] [argmode] argtype } [, ...])) [CASCADE | RESTRICT]];
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified function does not exist.
- **function\_name**  
Specifies the name of the function to be deleted.  
Value range: an existing function name
- **argmode**  
Specifies the parameter mode of the function.
- **argname**  
Specifies the parameter name of the function.
- **argtype**  
Specifies the parameter type of the function.
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the function.
  - **RESTRICT**: refuses to delete the function if any objects depend on it. This is the default action.

### Examples

See [Examples](#) in section "CREATE FUNCTION."

## Helpful Links

[ALTER FUNCTION](#) and [CREATE FUNCTION](#)

## 7.14.97 DROP GLOBAL CONFIGURATION

### Description

Drops parameter values from the `gs_global_config` system catalog.

### Precautions

- Only the initial database user can run this command.
- The parameter name cannot be **weak\_password** or **undostoragetype**.

### Syntax

```
DROP GLOBAL CONFIGURATION name [, ...];
```

### Parameters

- **name**  
The parameter must exist in the `gs_global_config` system catalog. If you delete a parameter that does not exist, an error will be reported.

### Helpful Links

[ALTER GLOBAL CONFIGURATION](#)

## 7.14.98 DROP GROUP

### Description

Drops a user group. **DROP GROUP** is an alias for **DROP ROLE**.

### Precautions

It is available only to users with the CREATE ROLE permission granted by the administrator.

### Syntax

```
DROP GROUP [IF EXISTS] group_name [, ...];
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified role does not exist.
- **group\_name**  
Specifies the name of the role to be deleted.  
Value range: an existing role name

### Helpful Links

- [CREATE GROUP](#)
- [ALTER GROUP](#)

- [DROP ROLE](#)

## 7.14.99 DROP INDEX

### Description

Drops an index.

### Precautions

Only the index owner, owner of the schema where the index resides, a user who has the INDEX permission on the table where the index resides, or a user who has the DROP ANY INDEX permission can run the **DROP INDEX** command. System administrators have this permission by default.

### Syntax

```
DROP INDEX [IF EXISTS]
index_name [, ...] [CASCADE | RESTRICT];
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified index does not exist.
- **index\_name**  
Specifies the name of the index to be deleted.  
Value range: an existing index
- **CASCADE | RESTRICT**
  - **CASCADE**: cascadingly deletes the objects that depend on the index.
  - **RESTRICT**: refuses to delete the index if any objects depend on it. This is the default action.

### Examples

See [Examples](#) in section "CREATE INDEX."

### Helpful Links

[ALTER INDEX](#) and [CREATE INDEX](#)

## 7.14.100 DROP LANGUAGE

This version does not support this syntax.

## 7.14.101 DROP MASKING POLICY

### Description

Deletes a masking policy.

## Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

## Syntax

```
DROP MASKING POLICY [IF EXISTS] policy_name;
```

## Parameters

- **policy\_name**  
Specifies the audit policy name, which must be unique.  
Value range: a string. It must comply with the [naming convention](#). The value is an existing policy name.

## Examples

```
-- Delete a masking policy.
gaussdb=# DROP MASKING POLICY IF EXISTS maskpol1;

-- Delete a group of anonymization policies.
gaussdb=# DROP MASKING POLICY IF EXISTS maskpol1, maskpol2, maskpol3;
```

## Helpful Links

[ALTER MASKING POLICY](#) and [CREATE MASKING POLICY](#)

## 7.14.102 DROP MATERIALIZED VIEW

### Function

**DROP MATERIALIZED VIEW** deletes an existing materialized view from the database.

### Precautions

The owner of a materialized view, owner of the schema of the materialized view, users granted with the DROP permission on the materialized view, or users granted with the DROP ANY TABLE permission can run the **DROP MATERIALIZED VIEW** command. By default, the system administrator has the permission to run the command.

### Syntax

```
DROP MATERIALIZED VIEW [IF EXISTS] mv_name [, ...] [CASCADE | RESTRICT];
```

### Parameter Description

- **IF EXISTS**  
Reports a notice instead of an error if the specified materialized view does not exist.
- **mv\_name**  
Name of the materialized view to be deleted.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on a materialized view.
  - **RESTRICT**: refuses to delete a materialized view if any objects depend on it. This is the default value.

## Examples

```
-- Create a table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int) WITH(STORAGE_TYPE=ASTORE);

-- Create a materialized view named my_mv.
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

-- Delete the materialized view named my_mv.
gaussdb=# DROP MATERIALIZED VIEW my_mv;

-- Delete the table.
gaussdb=# DROP TABLE my_table;
```

## Helpful Links

[ALTER MATERIALIZED VIEW](#), [CREATE INCREMENTAL MATERIALIZED VIEW](#), [CREATE MATERIALIZED VIEW](#), [CREATE TABLE](#), [REFRESH INCREMENTAL MATERIALIZED VIEW](#), and [REFRESH MATERIALIZED VIEW](#)

## 7.14.103 DROP MODEL

This syntax is not supported in distributed scenarios.

## 7.14.104 DROP NODE

### Description

Drops a node.

### Precautions

DROP NODE is an API of the cluster management tool. You are advised not to use this API, because doing so affects the cluster. Only the administrator has the permission to use this API.

### Syntax

```
DROP NODE [IF EXISTS] nodename [WITH (cnnodename [...])];
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified node does not exist.
- **nodename**  
Specifies the name of the node to be deleted.  
Value range: an existing node name
- **cnnodename**

Specifies the CN name. If it is defined, the command will be executed on this node in addition to the current connected CN. If it is not specified, DN deletion must be performed on all CNs, and CN deletion must be performed on all CNs except the CN to be deleted.

Value range: an existing CN name

## Examples

See [Examples](#) in "ALTER NODE."

## Helpful Links

- [CREATE NODE](#)
- [ALTER NODE](#)

## 7.14.105 DROP NODE GROUP

### Description

Deletes a node group.

### Precautions

- DROP NODE GROUP is an API of the cluster management tool.
- Only system administrators or a user who has the DROP permission can perform this operation.

### Syntax

```
DROP NODE GROUP groupname [DISTRIBUTE FROM src_group_name];
```

### Parameters

- **groupname**  
Specifies the name of the node group to be deleted.  
Value range: an existing node group name

## Examples

See [Examples](#) in "ALTER NODE GROUP."

## Helpful Links

[CREATE NODE GROUP](#)

## 7.14.106 DROP OWNED

### Description

Drops the database objects owned by a database role.

## Precautions

- This interface will revoke the role's permissions on all objects in the current database and shared objects (databases and tablespaces).
- **DROP OWNED** is often used to prepare for removing one or more roles. Because **DROP OWNED** affects only the objects in the current database, you need to run this statement in each database that contains the objects owned by the role to be removed.
- Using the **CASCADE** option may cause this statement to recursively remove objects owned by other users.
- The databases and tablespaces owned by the role will not be removed.
- Private database links owned by a role can be dropped only after the **CASCADE** option is added.

## Syntax

```
DROP OWNED BY name [, ...] [CASCADE | RESTRICT];
```

## Parameters

- **name**  
Specifies the role name.
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the objects to be deleted.
  - **RESTRICT**: refuses to delete the objects if other objects depend on them. This is the default action.

## Helpful Links

[REASSIGN OWNED](#) , [DROP ROLE](#)

## 7.14.107 DROP PROCEDURE

### Description

Drops a stored procedure.

### Syntax

```
DROP PROCEDURE [IF EXISTS] procedure_name ;
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified stored procedure does not exist.
- **procedure\_name**  
Specifies the name of the stored procedure to be deleted.  
Value range: an existing stored procedure name

## Examples

See [Examples](#) in "CREATE PROCEDURE."

## Helpful Links

[CREATE PROCEDURE](#)

## 7.14.108 DROP RESOURCE LABEL

### Description

Deletes a resource label.

### Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

### Syntax

```
DROP RESOURCE LABEL [IF EXISTS] label_name[, ...];
```

### Parameters

- **label\_name**  
Specifies the resource label name.  
Value range: a string. It must comply with the [naming convention](#).

### Examples

```
-- Delete a resource label.
gaussdb=# DROP RESOURCE LABEL IF EXISTS res_label1;

-- Delete a group of resource labels.
gaussdb=# DROP RESOURCE LABEL IF EXISTS res_label1, res_label2, res_label3;
```

## Helpful Links

[ALTER RESOURCE LABEL](#) and [CREATE RESOURCE LABEL](#)

## 7.14.109 DROP ROLE

### Description

Drops a role.

### Syntax

```
DROP ROLE [IF EXISTS] role_name [, ...];
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified role does not exist.

- **role\_name**  
Specifies the name of the role to be deleted.  
Value range: an existing role name

## Examples

See [Examples](#) in section "CREATE ROLE."

## Helpful Links

[CREATE ROLE](#), [ALTER ROLE](#), and [SET ROLE](#)

## 7.14.110 DROP ROW LEVEL SECURITY POLICY

### Description

Deletes a row-level security policy from a table.

### Precautions

Only the table owner or administrators can delete a row-level security policy from the table.

### Syntax

```
DROP [ROW LEVEL SECURITY] POLICY [IF EXISTS] policy_name ON table_name [CASCADE | RESTRICT]
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified row-level security policy does not exist.
- **policy\_name**  
Specifies the name of a row-level security policy to be deleted.
- **table\_name**  
Specifies the name of the table containing the row-level security policy.
- **CASCADE | RESTRICT**  
Currently, no objects depend on row-level security policies. Therefore, **CASCADE** is equivalent to **RESTRICT**, and they are reserved to ensure backward compatibility.

## Examples

```
-- Create the data table all_data.
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

-- Create a row-level security policy.
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

-- Delete the row-level security policy.
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

-- Delete the all_data table.
gaussdb=# DROP TABLE all_data;
```

## Helpful Links

[ALTER ROW LEVEL SECURITY POLICY](#) and [CREATE ROW LEVEL SECURITY POLICY](#)

## 7.14.111 DROP SCHEMA

### Description

Deletes a schema from the current database.

### Precautions

- Only the schema owner or a user granted with the DROP permission can run the **DROP SCHEMA** command. System administrators have this permission by default.
- Only the initial user and O&M administrators can run the **DROP SCHEMA** command for O&M administrators. Other users cannot run **DROP SCHEMA** for O&M administrators.
- You are not allowed to drop DBE\_PLDEVELOPER when **allow\_system\_table\_mods** is disabled.

### Syntax

```
DROP SCHEMA [IF EXISTS] schema_name [, ...] [CASCADE | RESTRICT];
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified schema does not exist.
- **schema\_name**  
Specifies the name of the schema to be deleted.  
Value range: an existing schema name
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes all the objects contained in the schema.
  - **RESTRICT**: refuses to delete the schema if the schema contains objects. This is the default action.

---

#### NOTICE

Schemas beginning with **pg\_temp** or **pg\_toast\_temp** are for internal use. Do not delete them. Otherwise, unexpected consequences may be incurred.

---

#### NOTE

The schema currently being used cannot be deleted. To delete it, switch to another schema first.

## Examples

See [Examples](#) in section "CREATE SCHEMA."

## Helpful Links

[ALTER SCHEMA](#) and [CREATE SCHEMA](#).

## 7.14.112 DROP SEQUENCE

### Description

Deletes a sequence from the current database.

### Precautions

Only the owner of a sequence, the owner of the schema to which the sequence belongs, or a user granted the DROP permission on a sequence or a user granted the DROP ANY SEQUENCE permission can delete a sequence. A system administrator has this permission by default.

### Syntax

```
DROP SEQUENCE [IF EXISTS] { [schema.] sequence_name } [, ...] [CASCADE | RESTRICT];
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified sequence does not exist.
- **sequence\_name**  
Specifies the name of the sequence to be deleted.
- **CASCADE | RESTRICT**
  - **CASCADE**: cascadingly deletes the objects that depend on the data server.
  - **RESTRICT**: refuses to delete the server if any objects depend on it. This is the default action.

### Examples

```
-- Create an ascending sequence named serial, starting from 101.
gaussdb=# CREATE SEQUENCE serial START 101;

-- Delete a sequence.
gaussdb=# DROP SEQUENCE serial;
```

## Helpful Links

[ALTER SEQUENCE](#) and [DROP SEQUENCE](#)

## 7.14.113 DROP SERVER

### Description

Deletes an existing data server.

## Precautions

Only the server owner or a user granted with the DROP permission can run the **DROP SERVER** command. System administrators have this permission by default.

## Syntax

```
DROP SERVER [IF EXISTS] server_name [CASCADE | RESTRICT] ;
```

## Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified data server does not exist.
- **server\_name**  
Specifies the name of the data server to be deleted.
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the data server.
  - **RESTRICT**: refuses to delete the server if any objects depend on it. This is the default action.

## Examples

```
-- Create a server.
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER

-- Drop my_server.
gaussdb=# DROP SERVER my_server;
DROP SERVER
```

## Helpful Links

[ALTER SERVER](#) and [CREATE SERVER](#)

## 7.14.114 DROP SYNONYM

### Function

**DROP SYNONYM** deletes a synonym.

### Precautions

Only the synonym owner or a user granted with the DROP ANY SYNONYM permission can run the **DROP SYNONYM** command. The system administrator has this permission by default.

### Syntax

```
DROP SYNONYM [IF EXISTS] synonym_name [CASCADE | RESTRICT] ;
```

### Parameter Description

- **IF EXISTS**

Reports a notice instead of an error if the specified synonym does not exist.

- **synonym\_name**  
Specifies the name (optionally schema-qualified) of the synonym to be deleted.
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects (such as views) that depend on the synonym.
  - **RESTRICT**: refuses to delete the synonym if any objects depend on it. This is the default action.

## Examples

See [Examples](#) in **CREATE SYNONYM**.

## Helpful Links

[ALTER SYNONYM](#) and [CREATE SYNONYM](#)

## 7.14.115 DROP TABLE

### Description

Deletes a table.

### Precautions

- After DROP TABLE deletes the table, the indexes depending on the table are deleted, and the functions and stored procedures that need to use this table cannot be executed. Deleting a partitioned table also deletes all partitions in the table.
- The owner of a table, the owner of the schema of the table, users granted with the DROP permission on the table, or users granted with the DROP ANY TABLE permission can delete the specified table. The system administrator has the permission to delete the specified table by default.

### Syntax

```
DROP TABLE [IF EXISTS]
{ [schema.]table_name } [, ...] [CASCADE | RESTRICT] [PURGE];
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified table does not exist.
- **schema**  
Specifies the schema name.
- **table\_name**  
Specifies the table name.
- **CASCADE | RESTRICT**

- **CASCADE**: allows to cascadingly delete the objects (such as views) that depend on the table.
- **RESTRICT**: refuses to delete the table if any objects depend on it. This is the default action.
- **PURGE**  
Specifies that even if the recycle bin function is enabled, the table is physically dropped instead of being moved to the recycle bin when you use DROP TABLE to delete tables.

## Examples

See [Examples](#) in section "CREATE TABLE."

## Helpful Links

[ALTER TABLE](#) and [CREATE TABLE](#)

## 7.14.116 DROP TABLESPACE

### Description

Deletes a tablespace.

### Precautions

- Only the tablespace owner or a user granted with the DROP permission can run the **DROP TABLESPACE** command. The system administrator has this permission by default.
- The tablespace to be deleted should not contain any database objects. Otherwise, an error will be reported.
- DROP TABLESPACE cannot be rolled back and therefore cannot be run in transaction blocks.
- During execution of DROP TABLESPACE, database queries by other sessions using **\db** may fail and need to be reattempted.
- If DROP TABLESPACE fails to be executed, execute DROP TABLESPACE IF EXISTS.

### Syntax

```
DROP TABLESPACE [IF EXISTS] tablespace_name;
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified tablespace does not exist.
- **tablespace\_name**  
Specifies the name of the tablespace to be deleted.  
Value range: an existing tablespace name

## Examples

See [Examples](#) in section "CREATE TABLESPACE."

## Helpful Links

[ALTER TABLESPACE](#) and [CREATE TABLESPACE](#)

## Suggestions

- **DROP TABLESPACE**  
Do not delete tablespaces during transactions.

## 7.14.117 DROP TRIGGER

### Function

**DROP TRIGGER** deletes a trigger.

### Precautions

Only the trigger owner or a user granted with the DROP ANY TRIGGER permission can run the **DROP TRIGGER** command. The system administrator has this permission by default.

### Syntax

```
DROP TRIGGER [IF EXISTS] trigger_name ON table_name [CASCADE | RESTRICT];
```

### Parameter Description

- **IF EXISTS**  
Reports a notice instead of an error if the specified trigger does not exist.
- **trigger\_name**  
Specifies the name of the trigger to be deleted.  
Value range: an existing trigger name
- **table\_name**  
Specifies the name of the table containing the trigger.  
Value range: name of the table containing the trigger
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the trigger.
  - **RESTRICT**: refuses to delete the trigger if any objects depend on it. This is the default action.

## Examples

For details, see [Examples](#) in [CREATE TRIGGER](#).

## Helpful Links

[CREATE TRIGGER](#), [ALTER TRIGGER](#), and [ALTER TABLE](#)

## 7.14.118 DROP TYPE

### Description

Drops a user-defined data type.

### Precautions

Only the owner of a type, a user granted the DROP permission on a type, or a user granted the DROP ANY TYPE permission on a sequence can run the **DROP TYPE** command. System administrators have this permission by default.

### Syntax

```
DROP TYPE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

### Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified type does not exist.
- **name**  
Specifies the name (optionally schema-qualified) of the type to be deleted.
- **CASCADE**  
Automatically deletes the objects (such as fields, functions, and operators) that depend on the type.  
**RESTRICT**  
Refuses to delete the type if any objects depend on it. This is the default action.

### Examples

See [Examples](#) in "CREATE TYPE."

### Helpful Links

[CREATE TYPE](#) and [ALTER TYPE](#)

## 7.14.119 DROP USER

### Description

Deletes a user and the schema with the same name as the user.

### Precautions

- **CASCADE** is used to delete the objects (excluding databases) that depend on the user. It cannot delete locked objects unless the objects are unlocked or the processes locking the objects are killed.
- If the dependent objects are other databases or reside in other databases, manually delete them before deleting the user from the current database. DROP USER cannot drop objects across databases.

- Before deleting a user, you need to delete all the objects owned by the user and revoke the user's permissions on other objects. Alternatively, you can specify **CASCADE** to delete the objects owned by the user and the granted permissions.
- If the user has an error table specified when the GDS foreign table is created, the user cannot be deleted by specifying the **CASCADE** keyword in the **DROP USER** statement.

## Syntax

```
DROP USER [IF EXISTS] user_name [, ...] [CASCADE | RESTRICT];
```

## Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified user does not exist.
- **user\_name**  
Specifies the name of the user to be deleted.  
Value range: an existing username
- **CASCADE | RESTRICT**
  - **CASCADE**: cascadingly deletes objects that depend on the user and revokes the permissions granted to the user.
  - **RESTRICT**: refuses to delete a user if the user has any dependent objects or has been granted permissions on other objects. This is the default value.

### NOTE

In GaussDB, the **postgresql.conf** file contains the **enable\_kill\_query** parameter. This parameter affects **CASCADE**.

- If **enable\_kill\_query** is **on** and **CASCADE** is used, the statement automatically kills the processes locking dependent objects and then deletes the specified user.
- If **enable\_kill\_query** is **off** and **CASCADE** is used, the statement waits until the processes locking dependent objects stop and then deletes the specified user.

## Examples

See [Examples](#) in section "CREATE USER."

## Helpful Links

[ALTER USER](#) and [CREATE USER](#)

## 7.14.120 DROP VIEW

### Description

**DROP VIEW** deletes a view from a database.

## Precautions

The owner of a view, owner of the schema of the view, users granted with the DROP permission on the view, or users granted with the DROP ANY TABLE permission can run the **DROP VIEW** command. By default, the system administrator has the permission to run the command.

## Syntax

```
DROP VIEW [IF EXISTS] view_name [...] [CASCADE | RESTRICT];
```

## Parameters

- **IF EXISTS**  
Reports a notice instead of an error if the specified view does not exist.
- **view\_name**  
Specifies the name of the view to be deleted.  
Value range: an existing view name
- **CASCADE | RESTRICT**
  - **CASCADE**: cascadingly deletes the objects (such as other views) that depend on the view.
  - **RESTRICT**: refuses to delete the view if any objects depend on it. This is the default action.

## Examples

See [Examples](#) in section "CREATE VIEW."

## Helpful Links

[ALTER VIEW](#) and [CREATE VIEW](#)

## 7.14.121 DROP WEAK PASSWORD DICTIONARY

### Description

Clears all weak passwords in gs\_global\_config.

### Precautions

Only the initial user and users with the SYSADMIN or CREATEROLE permission can execute this syntax.

### Syntax

```
DROP WEAK PASSWORD DICTIONARY;
```

### Example

See [CREATE WEAK PASSWORD DICTIONARY](#).

## Helpful Links

### [13.14.82-CREATE WEAK PASSWORD DICTIONARY](#)

## 7.14.122 EXECUTE

### Description

Executes a prepared statement. Because a prepared statement exists only in the lifetime of the session, the prepared statement must be created earlier in the current session by using the PREPARE statement.

### Precautions

If the PREPARE statement declares some parameters when the prepared statement is created, the parameter set passed to the EXECUTE statement must be compatible. Otherwise, an error occurs.

### Syntax

```
EXECUTE name [(parameter [, ...])];
```

### Parameters

- **name**  
Specifies the name of the prepared statement to be executed.
- **parameter**  
Specifies a parameter of the prepared statement. It must be an expression that generates a value compatible with the data type of the parameter specified when the prepared statement was created. **ROWNUM** cannot be used as a parameter.

### Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the reason table.
gaussdb=# CREATE TABLE tpcds.reason (
 CD_DEMO_SK INTEGER NOT NULL,
 CD_GENDER character(16) ,
 CD_MARITAL_STATUS character(100)
);

-- Insert data.
gaussdb=# INSERT INTO tpcds.reason VALUES(51, 'AAAAAAAADDAAAAAA', 'reason 51');

-- Create the reason_t1 table.
gaussdb=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

-- Create a prepared statement for an INSERT statement and execute the prepared statement.
gaussdb=# PREPARE insert_reason(integer,character(16),character(100)) AS INSERT INTO tpcds.reason_t1
VALUES($1,$2,$3);

gaussdb=# EXECUTE insert_reason(52, 'AAAAAAAADDAAAAAA', 'reason 52');

-- Delete the reason and reason_t1 tables.
gaussdb=# DROP TABLE tpcds.reason;
gaussdb=# DROP TABLE tpcds.reason_t1;
```

```
-- Delete the schema.
gaussdb=# DROP SCHEMA tpccs CASCADE;
```

## 7.14.123 EXECUTE DIRECT

### Description

Executes an SQL statement on a specified node. Generally, the cluster automatically allocates an SQL statement to proper nodes. EXECUTE DIRECT is mainly used for database maintenance and testing.

### Precautions

- When **enable\_nonsysadmin\_execute\_direct** is **off**, only a system administrator or monitor administrator has the EXECUTE DIRECT permission.
- To ensure data consistency across nodes, only the SELECT statement can be used. Transaction statements, DDL, and DML cannot be used.
- When the stddev aggregation calculation is performed on the specified DN using such statements, the result set is returned in triplet. For example, {3, 8, 30} indicates that the count result is 3, the sum result is 8, and the sum of squares is 30. When the AVG aggregation calculation is performed on the specified DN using such statements, the result set is returned in a binary tuple, for example, {4,2}. The result of count is 4, and that of sum is 2.
- When multiple nodes are specified, aggregate functions are not supported. If the query contains an aggregate function, the message "EXECUTE DIRECT on multinode not support agg functions." is returned.
- CN nodes do not store user table data. Therefore, do not execute SELECT for querying user tables on a CN.
- If the SQL statement to be executed is also EXECUTE DIRECT, do not nest it into EXECUTE DIRECT; instead, directly execute the inner EXECUTE DIRECT.
- The query result of the **agg** function is inconsistent with that on the CN. Multiple pieces of information are returned. The **array\_avg** function is not supported.
- If the **nodeoid** or **nodeoid list** parameter is specified, this parameter takes effect only when the **enable\_direct\_standby\_datanodes** parameter is successfully set in distributed mode. To query data on a specified DN, invalid or duplicate **nodeoid** values are not supported, and **nodeoid** and **nodename** cannot be used together.

### Syntax

```
EXECUTE DIRECT ON (nodename [, ...]) query ;
EXECUTE DIRECT ON (nodeoid [, ...]) query ;
EXECUTE DIRECT ON { COORDINATORS | DATANODES | ALL } query;
```

### Parameters

- **nodename**  
Specifies the node name.  
Value range: an existing node name

- **nodeoid**  
Node OID.  
Value range: an existing DN OID, which can be obtained from the **PGXC\_NODE** system catalog.
- **query**  
Specifies the SQL statement to be executed.
- **COORDINATORS**  
Run the query statement on all CNs.
- **DATANODES**  
Run the query statement on all DNs.
- **ALL**  
Run the query statement on all CNs and DNs.

## Examples

```
-- Query the node distribution status of the current cluster.
gaussdb=# SELECT * FROM pgxc_node;
 node_name | node_type | node_port | node_host | node_port1 | node_host1 | hostis_primary |
 nodeis_primary | nodeis_preferred | node_id | sctp_port | control_port | sctp_port1 | control_port1
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
cn_5001 | C | 8050 | 10.180.155.74 | 8050 | 10.180.155.74 | t | f | |
f | 1120683504 | 0 | 0 | 0 | 0 |
cn_5003 | C | 8050 | 10.180.157.130 | 8050 | 10.180.157.130 | t | f | |
f | -125853378 | 0 | 0 | 0 | 0 |
dn_6001_6002 | D | 40050 | 10.180.155.74 | 45050 | 10.146.187.231 | t | f | |
f | 1644780306 | 40052 | 40052 | 45052 | 45052 |
dn_6003_6004 | D | 40050 | 10.146.187.231 | 45050 | 10.180.157.130 | t | f | |
f | -966646068 | 40052 | 40052 | 45052 | 45052 |
dn_6005_6006 | D | 40050 | 10.180.157.130 | 45050 | 10.180.155.74 | t | f | |
f | 868850011 | 40052 | 40052 | 45052 | 45052 |
cn_5002 | C | 8050 | localhost | 8050 | localhost | t | f | |
-1736975100 | 0 | 0 | 0 | 0 |
(6 rows)

-- Query records in the tpcds.customer_address table on dn_6001_6002.
gaussdb=# EXECUTE DIRECT ON(dn_6001_6002) 'select count(*) from tpcds.customer_address';
 count

16922
(1 row)

-- Query records in the tpcds.customer_address table.
gaussdb=# SELECT count(*) FROM tpcds.customer_address;
 count

50000
(1 row)
```

## 7.14.124 EXPDP DATABASE

### Description

Exports all physical files in a database.

### Syntax

```
EXPDP DATABASE db_name LOCATION = 'directory';
```

## Parameters

- **db\_name**  
Name of the database to be exported.
- **directory**  
Directory for storing exported files.

## Example

```
expdp database test location = '/data1/expdp/database';
```

## 7.14.125 EXPDP TABLE

### Description

Exports all table-related index, sequence, partition, TOAST, and TOAST index files.

### Syntax

```
EXPDP TABLE table_name LOCATION = 'directory';
```

### Parameters

- **table\_name**  
Name of the table to be exported.
- **directory**  
Directory for storing exported files.

## Example

```
expdp table test_t location = '/data1/expdp/table0';
```

## 7.14.126 EXPLAIN

### Description

Shows the execution plan of an SQL statement.

The execution plan shows how the tables referenced by the statement will be scanned - by plain sequential scan, index scan, etc. - and if multiple tables are referenced, what join algorithms will be used to bring together the required rows from each input table.

The most critical part of the display is the estimated statement execution cost, which is the plan generator's guess at how long it will take to run the statement.

The **ANALYZE** option causes the statement to be actually executed, not only planned. The total elapsed time expended within each plan node (in milliseconds) and total number of rows it actually returned are added to the display. This is useful for determining whether the plan generator's estimated value is close to the actual one.

## Precautions

The statement is actually executed when ANALYZE is used. If you want to use EXPLAIN ANALYZE on an INSERT, UPDATE, DELETE, CREATE TABLE AS, or EXECUTE statement without letting the statement affect your data, use this approach:

```
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

## Syntax

- Display the execution plan of an SQL statement, which supports multiple options and has no requirements for the order of options.

```
EXPLAIN [(option [, ...])] statement;
```

The syntax of the **option** clause is as follows:

```
ANALYZE [boolean] |
ANALYZE [boolean] |
VERBOSE [boolean] |
COSTS [boolean] |
CPU [boolean] |
DETAIL [boolean] |
NODES [boolean] |
NUM_NODES [boolean] |
BUFFERS [boolean] |
TIMING [boolean] |
PLAN [boolean] |
BLOCKNAME [boolean] |
FORMAT { TEXT | XML | JSON | YAML }
```

- Display the execution plan of an SQL statement, where options are in order.

```
EXPLAIN { [ANALYZE | ANALYSE] [VERBOSE] | PERFORMANCE } statement;
```

## Parameters

- **statement**  
Specifies the SQL statement to explain.
- **ANALYZE boolean | ANALYSE boolean**  
Specifies whether to display actual run times and other statistics. When two parameters are used at the same time, the latter option takes effect.  
Value range:
  - **TRUE** (default): displays them.
  - **FALSE**: does not display them.
- **VERBOSE boolean**  
Specifies whether to display additional information regarding the plan.  
Value range:
  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.
- **COSTS boolean**  
Specifies whether to display the estimated total cost of each plan node, estimated number of rows, estimated width of each row.  
Value range:
  - **TRUE** (default): displays them.

- **FALSE**: does not display them.
- **CPU boolean**

Specifies whether to display CPU usage. It should be used together with the **ANALYZE** option.

Value range:

  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.
- **DETAIL boolean**

Specifies whether to display DN information. It should be used together with the **ANALYZE** option.

Value range:

  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.
- **NODES boolean**

Specifies whether to display information about the nodes executed by query.

Value range:

  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.
- **NUM\_NODES boolean**

Specifies whether to display the number of executing nodes.

Value range:

  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.
- **BUFFERS boolean**

Specifies whether to display buffer usage. It should be used together with the **ANALYZE** or **ANALYSE** option.

Value range:

  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.
- **TIMING boolean**

Specifies whether to display the actual startup time and time spent on the output node. It should be used together with the **ANALYZE** or **ANALYSE** option.

Value range:

  - **TRUE** (default): displays them.
  - **FALSE**: does not display them.
- **PLAN boolean**

Specifies whether to store the execution plan in **PLAN\_TABLE**. If this parameter is set to **on**, the execution plan is stored in **PLAN\_TABLE** and not displayed on the screen. Therefore, this parameter cannot be used together with other parameters when it is set to **on**.

Value range:

- **TRUE** (default): The execution plan is stored in **PLAN\_TABLE** and not displayed on the screen. If the plan is stored successfully, "EXPLAIN SUCCESS" is returned.
- **FALSE**: The execution plan is not stored but is printed on the screen.
- **BLOCKNAME boolean**

Specifies whether to display the query block where each operation of the plan is located. When this option is enabled, the name of the query block where each operation is performed is displayed in the **Query Block** column. This helps users obtain the query block name and use hints to modify the execution plan.

  - **TRUE** (default value): The name of the query block where each operation is located is displayed in the **Query Block** column. This option must be used in the pretty mode. For details, see [Hint Specifying the Query Block Where the Hint Is Located](#).
  - **FALSE**: The plan display is not affected.
- **FORMAT**

Specifies the output format.  
Value range: **TEXT**, **XML**, **JSON**, and **YAML**  
Default value: **TEXT**
- **PERFORMANCE**

Prints all relevant information in execution. Some information is described as follows:

  - **ex c/r**: indicates the average number of CPU cycles used by each row, which is equal to **(ex cyc) / (ex row)**.
  - **ex row**: indicates the number of executed rows.
  - **ex cyc**: indicates the number of used CPU cycles.
  - **inc cyc**: indicates the total number of CPU cycles used by subnodes.
  - **shared hit**: indicates the shared buffer hits of the operator.
  - **loops**: indicates the number of operator loop execution times.
  - **total\_calls**: indicates the total number of generated elements.
  - **remote query poll time stream gather**: indicates the operator used to listen to the network poll time when data on each DN reaches the CN.
  - **deserialize time**: indicates the time required for deserialization.
  - **estimated time**: indicates the estimated time.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer_address table.
gaussdb=# CREATE TABLE tpcds.customer_address(
ca_address_sk INTEGER NOT NULL,
ca_address_id CHARACTER(16) NOT NULL
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.customer_address VALUES (5000, 'AAAAAAAAABAAAAAAA'),(10000,
'AAAAAAAAACAAAAAAA');
```

```

-- Create the tpcds.customer_address_p1 table.
gaussdb=# CREATE TABLE tpcds.customer_address_p1 AS TABLE tpcds.customer_address;

-- Change the value of explain_perf_mode to normal.
gaussdb=# SET explain_perf_mode=normal;

-- Display an execution plan for simple queries in the table.
gaussdb=# EXPLAIN SELECT * FROM tpcds.customer_address_p1;
QUERY PLAN

Data Node Scan (cost=0.00..0.00 rows=0 width=0)
Node/s: All datanodes
(2 rows)

-- Generate an execution plan in JSON format (with explain_perf_mode being normal).
gaussdb=# EXPLAIN(FORMAT JSON) SELECT * FROM tpcds.customer_address_p1;
QUERY PLAN

[
 {
 "Plan": {
 "Node Type": "Data Node Scan",+
 "Startup Cost": 0.00, +
 "Total Cost": 0.00, +
 "Plan Rows": 0, +
 "Plan Width": 0, +
 "Node/s": "All datanodes" +
 }
 }
]
(1 row)

-- If there is an index and we use a query with an indexable WHERE condition, a different plan may be
displayed.
gaussdb=# EXPLAIN SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
QUERY PLAN

Data Node Scan (cost=0.00..0.00 rows=0 width=0)
Node/s: dn_6005_6006
(2 rows)

-- Generate an execution plan in YAML format (with explain_perf_mode being normal).
gaussdb=# EXPLAIN(FORMAT YAML) SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
QUERY PLAN

- Plan:
 Node Type: "Data Node Scan"+
 Startup Cost: 0.00 +
 Total Cost: 0.00 +
 Plan Rows: 0 +
 Plan Width: 0 +
 Node/s: "dn_6005_6006"
(1 row)

-- Suppress the execution plan of cost estimation.
gaussdb=# EXPLAIN(COSTS FALSE)SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
QUERY PLAN

Data Node Scan
Node/s: dn_6005_6006
(2 rows)

-- Generate an execution plan with aggregate functions for a query.
gaussdb=# EXPLAIN SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE
ca_address_sk<10000;
QUERY PLAN

```

```
Aggregate (cost=18.19..14.32 rows=1 width=4)
-> Streaming (type: GATHER) (cost=18.19..14.32 rows=3 width=4)
 Node/s: All datanodes
 -> Aggregate (cost=14.19..14.20 rows=3 width=4)
 -> Seq Scan on customer_address_p1 (cost=0.00..14.18 rows=10 width=4)
 Filter: (ca_address_sk < 10000)
(6 rows)

-- Delete the tpcds.customer_address_p1 table.
gaussdb=# DROP TABLE tpcds.customer_address_p1;

-- Delete the tpcds.customer_address table.
gaussdb=# DROP TABLE tpcds.customer_address;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Helpful Links

[ANALYZE | ANALYSE](#)

## 7.14.127 EXPLAIN PLAN

### Description

**EXPLAIN PLAN** saves information about an execution plan into the **PLAN\_TABLE** table. Different from the **EXPLAIN** statement, **EXPLAIN PLAN** only saves plan information; it does not print information on the screen.

### Syntax

```
EXPLAIN PLAN
[SET STATEMENT_ID = name]
FOR statement ;
```

### Parameters

- **EXPLAIN PLAN**  
**PLAN** saves plan information into **PLAN\_TABLE**. If information is stored successfully, "EXPLAIN SUCCESS" is returned.
- **STATEMENT\_ID**  
**STATEMENT\_ID** tags each query. The tag information will be stored in **PLAN\_TABLE**.
- **name**  
Specifies a query tag.  
Value range: a string.

#### NOTE

If **STATEMENT\_ID** is not set when the **EXPLAIN PLAN** statement is executed, **STATEMENT\_ID** is left empty by default. In addition, the value of **STATEMENT\_ID** cannot exceed 30 bytes. Otherwise, an error will be reported.

### Precautions

- **EXPLAIN PLAN** cannot be executed on DNs.
- Plan information cannot be collected for SQL statements that failed to be executed.

- Data in **PLAN\_TABLE** is in a session-level lifecycle. Sessions are isolated from users, and therefore users can only view the data of the current session and current user.
- **PLAN\_TABLE** cannot be joined with GDS foreign tables.
- For a query that cannot be pushed down, object information cannot be collected and only such information as **REMOTE\_QUERY** and **CTE** can be collected. For details, see [Example 2](#).

## Example 1

You can perform the following steps to collect execution plans of SQL statements by running **EXPLAIN PLAN**:

### Step 1 Run **EXPLAIN PLAN**.

#### NOTE

After the **EXPLAIN PLAN** statement is executed, plan information is automatically stored in **PLAN\_TABLE**. **INSERT**, **UPDATE**, and **ANALYZE** cannot be performed on **PLAN\_TABLE**.

For details about **PLAN\_TABLE**, see [PLAN\\_TABLE](#).

```
-- Create tables foo1 and foo2.
gaussdb=# CREATE TABLE foo1(f1 int, f2 text, f3 text[]);
gaussdb=# CREATE TABLE foo2(f1 int, f2 text, f3 text[]);

-- Run EXPLAIN PLAN.
gaussdb=# EXPLAIN PLAN SET STATEMENT_ID = 'TPCH-Q4' FOR SELECT f1, count(*) FROM foo1 WHERE f1
> 1 AND f1 < 3 AND EXISTS (SELECT * FROM foo2) GROUP BY f1;
```

### Step 2 Query **PLAN\_TABLE**.

```
gaussdb=# SELECT * FROM plan_table;
```

### Step 3 Delete data from **PLAN\_TABLE**.

```
gaussdb=# DELETE FROM plan_table WHERE STATEMENT_ID = 'TPCH-Q4';
gaussdb=# DROP TABLE foo1;
gaussdb=# DROP TABLE foo2;
```

----End

## Example 2

For a query that cannot be pushed down, only such information as **REMOTE\_QUERY** and **CTE** can be collected from **PLAN\_TABLE** after **EXPLAIN PLAN** is executed.

Scenario 1: The optimizer generates a plan for pushing down statements. In this case, only **REMOTE\_QUERY** can be collected.

```
-- Create tables pt_t1 and pg_t2.
gaussdb=# CREATE TABLE pt_t1(a integer, b int, c int)WITH(autovacuum_enabled = off) DISTRIBUTE
hash(c);
gaussdb=# CREATE TABLE pt_t1(a int, b int, c int)WITH(autovacuum_enabled = off) DISTRIBUTE hash(c);
gaussdb=# EXPLAIN PLAN SET statement_id = 'test remote query' FOR SELECT current_user FROM pt_t1,
pt_t2;

-- Query PLAN_TABLE.
gaussdb=# SELECT * FROM plan_table;

-- Drop tables pt_t1 and pg_t2.
gaussdb=# DROP TABLE pt_t1;
gaussdb=# DROP TABLE pg_t2;
```

Query **PLAN\_TABLE**.

```
gaussdb=# SELECT * FROM plan_table;
```

Scenario 2: For a query with **WITH RECURSIVE** that cannot be pushed down, only **CTE** can be collected.

Disable **enable\_stream\_recursive** so that the query cannot be pushed down.

```
gaussdb=# SET enable_stream_recursive = off;
```

Run the **EXPLAIN PLAN** statement.

```
-- Create the chinamap table.
gaussdb=# CREATE TABLE chinamap
(
 id integer,
 pid integer,
 name text
) DISTRIBUTE BY hash(id);

-- Plan collected by plan_table.
gaussdb=# EXPLAIN PLAN SET statement_id = 'cte can not be push down'
FOR
WITH RECURSIVE rq AS
(
 SELECT id, name FROM chinamap WHERE id = 11
 UNION ALL
 SELECT origin.id, rq.name || ' > ' || origin.name
 FROM rq JOIN chinamap origin ON origin.pid = rq.id
)
SELECT id, name FROM rq ORDER BY 1;

-- Query PLAN_TABLE.
gaussdb=# SELECT * FROM plan_table;

-- Drop the chinamap table.
gaussdb=# DROP TABLE chinamap;
```

## 7.14.128 FETCH

### Description

**FETCH** retrieves rows using a previously created cursor.

A cursor has an associated position, which is used by **FETCH**. The cursor position can be before the first row of the query result, on any particular row of the result, or after the last row of the result.

- When created, a cursor is positioned before the first row.
- After fetching some rows, the cursor is positioned on the row most recently retrieved.
- If **FETCH** runs off the end of the available rows then the cursor is left positioned after the last row, or before the first row if fetching backward.
- **FETCH ALL** or **FETCH BACKWARD ALL** will always leave the cursor positioned after the last row or before the first row.

### Precautions

- If the cursor is declared with **NO SCROLL**, backward fetches like **FETCH BACKWARD** are not allowed.
- The forms **NEXT**, **PRIOR**, **FIRST**, **LAST**, **ABSOLUTE**, and **RELATIVE** fetch a single row after moving the cursor appropriately. If there is no such row, an empty result is returned, and the cursor is left positioned before the first row (backward fetch) or after the last row (forward fetch) as appropriate.

- The forms using FORWARD and BACKWARD retrieve the indicated number of rows moving in the forward or backward direction, leaving the cursor positioned on the last-returned row or after (backward fetch)/before (forward fetch) all rows if **count** exceeds the number of rows available.
- **RELATIVE 0**, **FORWARD 0**, and **BACKWARD 0** all request fetching the current row without moving the cursor, that is, re-fetching the most recently fetched row. This action will succeed unless the cursor is positioned before the first row or after the last row. If the cursor is positioned before the first row or after the last row, no row is returned.
- If the cursor of **FETCH** involves a non-system catalog, backward fetches like **BACKWARD**, **PRIOR**, and **FIRST** are not allowed.

## Syntax

```
FETCH [direction { FROM | IN }] cursor_name;
```

The **direction** clause specifies optional parameters.

```
NEXT
| PRIOR
| FIRST
| LAST
| ABSOLUTE count
| RELATIVE count
| count
| ALL
| FORWARD
| FORWARD count
| FORWARD ALL
| BACKWARD
| BACKWARD count
| BACKWARD ALL
```

## Parameters

- **direction**  
Defines the fetch direction.  
Value range:
  - **NEXT** (default value)  
Fetches the next row.
  - **PRIOR**  
Fetches the prior row.
  - **FIRST**  
Fetches the first row of the query (same as **ABSOLUTE 1**).
  - **LAST**  
Fetches the last row of the query (same as **ABSOLUTE -1**).
  - **ABSOLUTE** *count*  
Fetches the *count*th row of the query.  
**ABSOLUTE** fetches are not any faster than navigating to the desired row with a relative move: the underlying implementation must traverse all the intermediate rows anyway.  
Value range: a possibly-signed integer

- If *count* is positive, the *count*<sup>th</sup> row of the query will be fetched. If *count* is less than the current cursor position, rewind is required, which is currently not supported.
  - If *count* is negative or 0, backward scanning is required, which is currently not supported.
- RELATIVE count  
Fetches the *count*<sup>th</sup> succeeding row or the *count*<sup>th</sup> prior row if count is negative.  
Value range: a possibly-signed integer
  - If *count* is positive, the *count*<sup>th</sup> succeeding row will be fetched.
  - If *count* is negative or 0, backward scanning is required, which is currently not supported.
  - If the current row contains no data, **RELATIVE 0** returns null.
- count  
Fetches the next *count* rows (same as **FORWARD count**).
- ALL  
Fetches all remaining rows (same as **FORWARD ALL**).
- FORWARD  
Fetches the next row (same as **NEXT**).
- FORWARD count  
Fetches the next or prior *count* rows (same as **RELATIVE count**).
- FORWARD ALL  
Fetches all remaining rows.
- BACKWARD  
Fetches the prior row (same as **PRIOR**).
- BACKWARD count  
Fetches the prior *count* rows (scanning backwards).  
Value range: a possibly-signed integer
  - If *count* is positive, *count* prior rows will be fetched.
  - If *count* is a negative, *count* succeeding rows will be fetched.
  - **BACKWARD 0** re-fetches the current row, if any.
- BACKWARD ALL  
Fetches all prior rows (scanning backwards).
- **{ FROM | IN } cursor\_name**  
Specifies the cursor name using the keyword **FROM** or **IN**.  
Value range: an existing cursor name

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;
```

```
-- Create the tpcds.customer_address table.
gaussdb=# CREATE TABLE tpcds.customer_address
(
ca_address_sk INTEGER NOT NULL,
ca_address_id CHARACTER(16) NOT NULL,
ca_street_number INTEGER
ca_street_name CHARACTER (20)
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.customer_address VALUES (1, 'AAAAAAAAABAAAAAAA', '18', 'Jackson'),(2,
'AAAAAAAAACAAAAAAA', '362', 'Washington 6th'),(3, 'AAAAAAAADAAAAAAA', '585', 'Dogwood Washington');

-- (For the SELECT statement, traverse a table using a cursor.) Start a transaction.
gaussdb=# START TRANSACTION;

-- Set up cursor1.
gaussdb=# CURSOR cursor1 FOR SELECT * FROM tpcds.customer_address ORDER BY 1;

-- Fetch the first three rows in cursor1.
gaussdb=# FETCH FORWARD 3 FROM cursor1;
ca_address_sk | ca_address_id | ca_street_number | ca_street_name
-----+-----+-----+-----
 1 | AAAAAAAAABAAAAAAA | 18 | Jackson
 2 | AAAAAAACAACAAAAAAA | 362 | Washington 6th
 3 | AAAAAAADAAAAAAA | 585 | Dogwood Washington
(3 rows)

-- Close the cursor and commit the transaction.
gaussdb=# CLOSE cursor1;

-- End the transaction.
gaussdb=# END;

-- (For the VALUES clause, traverse the clause using a cursor.) Start a transaction.
gaussdb=# START TRANSACTION;

-- Set up cursor2.
gaussdb=# CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;

-- Fetch the first two rows in cursor2.
gaussdb=# FETCH FORWARD 2 FROM cursor2;
column1 | column2
-----+-----
0 | 3
1 | 2
(2 rows)

-- Close the cursor and commit the transaction.
gaussdb=# CLOSE cursor2;

-- End the transaction.
gaussdb=# END;

-- (WITH HOLD cursor) Start a transaction.
gaussdb=# START TRANSACTION;

-- Set up a WITH HOLD cursor.
gaussdb=# DECLARE cursor1 CURSOR WITH HOLD FOR SELECT * FROM tpcds.customer_address ORDER BY
1;

-- Fetch the first two rows in cursor1.
gaussdb=# FETCH FORWARD 2 FROM cursor1;
ca_address_sk | ca_address_id | ca_street_number | ca_street_name
-----+-----+-----+-----
 1 | AAAAAAAAABAAAAAAA | 18 | Jackson
 2 | AAAAAAACAACAAAAAAA | 362 | Washington 6th
(2 rows)
```

```
-- End the transaction.
gaussdb=# END;

-- Fetch the next row in cursor1.
gaussdb=# FETCH FORWARD 1 FROM cursor1;
 ca_address_sk | ca_address_id | ca_street_number | ca_street_name
-----+-----+-----+-----
 3 | AAAAAAAAAADAAAAA | 585 | Dogwood Washington
(1 row)

-- Close the cursor.
gaussdb=# CLOSE cursor1;

-- Delete the tpcds.customer_address table.
gaussdb=# DROP TABLE tpcds.customer_address;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Helpful Links

[CLOSE](#) and [MOVE](#)

## 7.14.129 GRANT

### Description

Grants permissions to roles and users.

GRANT is used in the following scenarios:

- **Granting system permissions to roles or users**

System permissions are also called user attributes, including SYSADMIN, CREATEDB, CREATEROLE, AUDITADMIN, MONADMIN, OPRADMIN, POLADMIN, INHERIT, REPLICATION, VCADMIN, and LOGIN.

They can be specified only by the CREATE ROLE or ALTER ROLE statement. The SYSADMIN permissions can be granted and revoked using GRANT ALL PRIVILEGE and REVOKE ALL PRIVILEGE, respectively. System permissions cannot be inherited by a user from a role, and cannot be granted using **PUBLIC**.

- **Granting database object permissions to roles or users**

Grant permissions on a database object (table, view, column, database, function, schema, or tablespace) to a role or user.

**GRANT** gives specific permissions on a database object to one or more roles. These permissions are added to those already granted, if any.

The keyword **PUBLIC** indicates that the permissions are to be granted to all roles, including those that might be created later. **PUBLIC** can be thought of as an implicitly defined group that always includes all roles. Any particular role will have the sum of permissions granted directly to it, permissions granted to any role it is presently a member of, and permissions granted to **PUBLIC**.

If **WITH GRANT OPTION** is specified, the recipient of the permission can in turn grant it to others. Without a grant option, the recipient cannot do that. This option cannot be granted to **PUBLIC**, which is a unique GaussDB attribute.

GaussDB grants the permissions on certain types of objects to **PUBLIC**. By default, permissions on tables, columns, sequences, foreign data sources, foreign servers, schemas, and tablespaces are not granted to **PUBLIC**, but the following permissions are granted to **PUBLIC**: **CONNECT** and **CREATE TEMP TABLE** permissions on databases, **EXECUTE** permission on functions, and **USAGE** permission on languages and data types (including domains). An object owner can revoke the default permissions granted to **PUBLIC** and grant permissions to other users as needed. For security purposes, you are advised to create an object and set its permissions in the same transaction so that other users do not have time windows to use the object. These default permissions can be modified using the **ALTER DEFAULT PRIVILEGES** command.

By default, an object owner has all permissions on the object. For security purposes, the owner can discard some permissions. However, the **ALTER**, **DROP**, **COMMENT**, **INDEX**, **VACUUM**, and re-grantable permissions of the object are inherent permissions implicitly owned by the owner.

- **Granting the permissions of one role or user to others**

Grant the permissions of one role or user to others. In this case, every role or user can be regarded as a set of one or more database permissions.

If **WITH ADMIN OPTION** is specified, the recipients can in turn grant the permissions to other roles or users or revoke the permissions they have granted to other roles or users. If recipients' permissions are changed or revoked later, the grantees' permissions will also change.

Database administrators can grant or revoke permissions for any roles or users. Roles with the **CREATEROLE** permission can grant or revoke permissions for non-admin roles.

- **Granting ANY permissions to roles or users**

Grant ANY permissions to a specified role or user. For details about the value range of the ANY permissions, see the syntax. If **WITH ADMIN OPTION** is specified, the grantee can grant the ANY permissions to or revoke them from other roles or users. The ANY permissions can be inherited by a role but cannot be granted to **PUBLIC**. When separation of duties is disabled, an initial user and SYSADMIN can grant the ANY permissions to or revoke them from any role or user.

Currently, the following ANY permissions are supported: CREATE ANY TABLE, ALTER ANY TABLE, DROP ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, CREATE ANY SEQUENCE, CREATE ANY INDEX, CREATE ANY FUNCTION, EXECUTE ANY FUNCTION, CREATE ANY TYPE, ALTER ANY TYPE, DROP ANY TYPE, ALTER ANY SEQUENCE, DROP ANY SEQUENCE, SELECT ANY SEQUENCE, ALTER ANY INDEX, DROP ANY INDEX, CREATE ANY SYNONYM, DROP ANY SYNONYM, CREATE ANY TRIGGER, ALTER ANY TRIGGER, and DROP ANY TRIGGER. For details about the ANY permission scope, see [Table 7-166](#).

## Precautions

- It is not allowed to grant the ANY permissions to **PUBLIC** or revoke the ANY permissions from **PUBLIC**.
- The ANY permissions are database permissions and are valid only for database objects that are granted with the permissions. For example, **SELECT ANY TABLE** only allows a user to view all user table data in the current

database, but the user does not have the permission to view user tables in other databases.

- The ANY permissions and the original permissions do not affect each other.
- If a user is granted with the **CREATE ANY TABLE** permission, the owner of a table created in a schema with the same name as the user is the creator of the schema. When the user performs other operations on the table, the user needs to be granted with the corresponding operation permission. Similarly, if a user is granted with the CREATE ANY FUNCTION, CREATE ANY TYPE, CREATE ANY SEQUENCE, or CREATE ANY INDEX permission, the owner of an object created in a schema with the same name is the owner of the schema. If a user is granted with the CREATE ANY TRIGGER or CREATE ANY SYNONYM permission, the owner of an object created in a schema with the same name is the creator.
- Exercise caution when granting the CREATE ANY FUNCTION permission to users to prevent other users from using DEFINER functions for privilege escalation.
- When GRANT is used to grant a user the permission to use a table, if the permission is not properly used, ALTER may be used to add expressions to the default values and constraints of the table, or indexes may be created to add expressions to INDEX. In this case, the permission may be exploited.
- When GRANT is used to grant the TRIGGER permission, if the permission is not properly used, the WHEN condition may be used to create expressions. When the trigger is triggered, the permission may be exploited.

## Syntax

- Grant the table or view access permission to a user or role.  

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ALTER | DROP
| COMMENT | INDEX | VACUUM } [, ...]
| ALL [PRIVILEGES] }
ON { [TABLE] table_name [, ...]
| ALL TABLES IN SCHEMA schema_name [, ...] }
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```
- Grant the column access permission to a user or role.  

```
GRANT { { SELECT | INSERT | UPDATE | REFERENCES | COMMENT } (column_name [, ...]) } [, ...]
| ALL [PRIVILEGES] (column_name [, ...]) }
ON [TABLE] table_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```
- Grant the sequence access permission to a specified user or role.  

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT } [, ...]
| ALL [PRIVILEGES] }
ON { [SEQUENCE] sequence_name [, ...]
| ALL SEQUENCES IN SCHEMA schema_name [, ...] }
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```
- Grant the database access permission to a user or role.  

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
| ALL [PRIVILEGES] }
ON DATABASE database_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```
- Grant the domain access permission to a user or role.  

```
GRANT { USAGE | ALL [PRIVILEGES] }
ON DOMAIN domain_name [, ...]
```

```
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

#### NOTE

In the current version, the domain access permission cannot be granted.

- Grant the CMK access permission to a specified user or role.

```
GRANT { { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
ON { CLIENT_MASTER_KEY client_master_key
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the column encryption key (CEK) access permission to a specified user or role.

```
GRANT { { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
ON { COLUMN_ENCRYPTION_KEY column_encryption_key
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the foreign data source access permission to a user or role.

```
GRANT { USAGE | ALL [PRIVILEGES] }
ON FOREIGN DATA WRAPPER fdw_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the foreign server access permission to a user or role.

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON FOREIGN SERVER server_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the function access permission to a user or role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON { FUNCTION {function_name ([{ [argmode] [arg_name] arg_type } [, ...]) } [, ...]
| ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the procedural procedure access permission to a user or role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON PROCEDURE {proc_name ([{ [argmode] [arg_name] arg_type } [, ...]) } [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the sub-cluster access permission to a user or role.

```
GRANT { { CREATE | USAGE | COMPUTE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
ON NODE GROUP group_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

#### NOTE

When the **create** permission of a sub-cluster is granted to a specified user or role, the **usage** and **compute** permissions are granted to the specified user or role by default.

- Grant the schema access permission to a user or role.

```
GRANT { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON SCHEMA schema_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

#### NOTE

When you grant table or view permissions to other users, you also need to grant the **USAGE** permission on the schema that the tables and views belong to. Without the **USAGE** permission, the users with table or view permissions can only see the object names, but cannot access them.

- Grant the large object access permission to a specified user or role.

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [PRIVILEGES] }
ON LARGE OBJECT loid [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

**NOTE**

In the current version, the large object access permission cannot be granted.

- Grant the tablespace access permission to a user or role.  
GRANT { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }  
ON TABLESPACE tablespace\_name [, ...]  
TO { [ GROUP ] role\_name | PUBLIC } [, ...]  
[ WITH GRANT OPTION ];

- Grant the type access permission to a user or role.  
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }  
ON TYPE type\_name [, ...]  
TO { [ GROUP ] role\_name | PUBLIC } [, ...]  
[ WITH GRANT OPTION ];

**NOTE**

In the current version, the type access permission cannot be granted.

- Grant the directory permission to a role.  
GRANT { { READ | WRITE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }  
ON DIRECTORY directory\_name [, ...]  
TO { [GROUP] role\_name | PUBLIC } [, ...]  
[WITH GRANT OPTION];
- Grant a role's permissions to another user or role.  
GRANT role\_name [, ...]  
TO role\_name [, ...]  
[ WITH ADMIN OPTION ];
- Grant the SYSADMIN permission to a role.  
GRANT ALL { PRIVILEGES | PRIVILEGE }  
TO role\_name;
- Grant the ANY permissions to another user or role.  
GRANT { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE |  
INSERT ANY TABLE | UPDATE ANY TABLE | DELETE ANY TABLE | CREATE ANY SEQUENCE |  
CREATE ANY INDEX | CREATE ANY FUNCTION | EXECUTE ANY FUNCTION | CREATE ANY TYPE |  
ALTER ANY TYPE | DROP ANY TYPE |  
ALTER ANY SEQUENCE | DROP ANY SEQUENCE | SELECT ANY SEQUENCE | ALTER ANY INDEX |  
DROP ANY INDEX | CREATE ANY SYNONYM | DROP ANY SYNONYM | CREATE ANY TRIGGER | ALTER  
ANY TRIGGER |  
DROP ANY TRIGGER } [, ...]  
TO [ GROUP ] role\_name [, ...]  
[ WITH ADMIN OPTION ];
- Grant the database link object permission to a specified user.  
GRANT { CREATE | ALTER | DROP } [PUBLIC] DATABASE LINK TO role\_name;

**NOTE**

- **PUBLIC:** creates a public database link visible to all users. If this clause is omitted, the database link is private and used only as a compatible API. The data that can be accessed on the remote database depends on the identity used by the database link during connection.
- When the permission to create a database link is granted to a user, the user can remotely access a database by using the IP address of the remote database. Exercise caution when granting this permission to users.
- In addition to the statement for directly granting the database link permission, you can also obtain the database link permission by inheriting permission and granting permission to an administrator.
- For details about database links, see [DATABASE LINK](#).

## Parameters

The possible permissions are:

- **SELECT**  
Allows **SELECT** from any table, view, or sequence. The **SELECT** permission on the corresponding column is also required for **UPDATE** or **DELETE**.
- **INSERT**  
Allows **INSERT** of a new row into a table.
- **UPDATE**  
Allows you to run the **UPDATE** statement on any column in the specified table. The **UPDATE** statement also requires the **SELECT** permission to query which rows need to be updated. **SELECT ... FOR UPDATE** and **SELECT ... FOR SHARE** also require this permission on at least one column, in addition to the **SELECT** permission.
- **DELETE**  
Allows you to run the **DELETE** statement to delete data from a specified table. The **DELETE** statement also requires the **SELECT** permission to query which rows need to be deleted.
- **TRUNCATE**  
Allows **TRUNCATE** on a table.
- **REFERENCES**  
Allows to create a foreign key constraint referencing a table. This permission is required on both referencing and referenced tables. This function is not supported in distributed mode.
- **TRIGGER**  
Allows the creation of a trigger on the specified table.
- **CREATE**
  - For databases, allows new schemas to be created within the database.
  - For schemas, allows new objects to be created within the schema. To rename an existing object, you must own the object and have the **CREATE** permission on the schema of the object.
  - For tablespaces, allows tables to be created within the tablespace, and allows databases and schemas to be created that have the tablespace as their default tablespace.
  - For sub-clusters, allows tables to be created within the sub-cluster.
- **CONNECT**  
Allows the grantee to connect to the database.
- **EXECUTE**  
Allows calling a function, including use of any operators that are implemented on top of the function.
- **USAGE**
  - For procedural languages, allows use of the language for the creation of functions in that language.
  - For schemas, allows access to objects contained in the schema. Without this permission, it is still possible to see the object names.

- For sequences, allows use of the **nextval** function.
- For sub-clusters, allows users who can access objects contained in the schema to access tables in the sub-cluster.
- For a key object, **USAGE** is the permission to use the key.
- **COMPUTE**  
For computing sub-clusters, allows users to perform elastic computing in the sub-cluster that they have the compute permission on.
- **ALTER**  
Allows users to modify the attributes of a specified object, excluding the owner and schema of the object.
- **DROP**  
Allows users to delete specified objects.
- **COMMENT**  
Allows users to define or modify comments of a specified object.
- **INDEX**  
Allows users to create indexes on specified tables, manage indexes on the tables, and perform REINDEX and CLUSTER operations on the tables.
- **VACUUM**  
Allows users to perform ANALYZE and VACUUM operations on specified tables.
- **ALL PRIVILEGES**  
Grants all available permissions to a user or role at a time. Only a system administrator has the **GRANT ALL PRIVILEGES** permission.

**GRANT** parameters are as follows:

- **role\_name**  
Specifies the username.
- **table\_name**  
Specifies the table name.
- **column\_name**  
Specifies the column name.
- **schema\_name**  
Specifies the schema name.
- **database\_name**  
Specifies the database name.
- **function\_name**  
Specifies the function name.
- **sequence\_name**  
Specifies the sequence name.
- **domain\_name**  
Specifies the domain type name.
- **fdw\_name**  
Specifies the foreign data wrapper name.

- **lang\_name**  
Specifies the language name.
- **type\_name**  
Specifies the type name.
- **group\_name**  
Specifies the sub-cluster name.
- **argmode**  
Specifies the parameter mode.  
Value range: a string. It must comply with the [naming convention](#).
- **arg\_name**  
Specifies the parameter name.  
Value range: a string. It must comply with the [naming convention](#).
- **arg\_type**  
Specifies the parameter type.  
Value range: a string. It must comply with the [naming convention](#).
- **loid**  
Specifies the identifier of the large object that includes this page.  
Value range: a string. It must comply with the [naming convention](#).
- **tablespace\_name**  
Specifies the tablespace name.
- **client\_master\_key**  
Name of the CMK.  
Value range: a string. It must comply with the [naming convention](#).
- **column\_encryption\_key**  
Name of the column encryption key.  
Value range: a string. It must comply with the [naming convention](#).
- **directory\_name**  
Specifies the directory name.  
Value range: a string. It must comply with the [naming convention](#).
- **WITH GRANT OPTION**  
If **WITH GRANT OPTION** is specified, the recipient of the permission can in turn grant it to others. Without a grant option, the recipient cannot do that. Grant options cannot be granted to **PUBLIC**.

When a non-owner of an object attempts to grant permissions on the object:

- The statement will fail outright if the user has no permissions whatsoever on the object.
- As long as some permission is available, the statement will proceed, but it will grant only those permissions for which the user has grant options.
- The **GRANT ALL PRIVILEGES** forms will issue a warning message if no grant options are held, while the other forms will issue a warning if grant options for any of the permissions specifically named in the statement are not held.

 **NOTE**

Database administrators can access all objects, regardless of object permission settings. It is unwise to operate as a system administrator except when necessary.

- **WITH ADMIN OPTION**

If **WITH ADMIN OPTION** is specified for a role, the grantee can grant the role to other roles or users or revoke the role from other roles or users.

For the ANY permissions, if **WITH ADMIN OPTION** is specified, the grantee can grant the ANY permissions to or revoke them from other roles or users.

**Table 7-166** ANY permissions

| Permission           | Description                                                                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CREATE ANY TABLE     | Users can create tables or views in the public and user schemas. The users must be granted with the permission to create sequences to create a table that contains serial columns.                                                                                             |
| ALTER ANY TABLE      | Users' <b>ALTER</b> permission on tables or views in the public and user schemas. If the users want to modify the unique index of a table to add a primary key constraint or unique constraint to the table, the users must be granted with the index permission on the table. |
| DROP ANY TABLE       | Users' <b>DROP</b> permission on tables or views in the public and user schemas.                                                                                                                                                                                               |
| SELECT ANY TABLE     | Users' <b>SELECT</b> permission on tables or views in the public and user schemas, which is still subject to row-level security.                                                                                                                                               |
| UPDATE ANY TABLE     | Users' <b>UPDATE</b> permission on tables or views in the public and user schemas, which is still subject to row-level security.                                                                                                                                               |
| INSERT ANY TABLE     | Users' <b>INSERT</b> permission on tables or views in the public and user schemas.                                                                                                                                                                                             |
| DELETE ANY TABLE     | Users' <b>DELETE</b> permission on tables or views in the public and user schemas, which is still subject to row-level security.                                                                                                                                               |
| CREATE ANY FUNCTION  | Users can create functions or stored procedures in the user schemas.                                                                                                                                                                                                           |
| EXECUTE ANY FUNCTION | Users' <b>EXECUTE</b> permission on functions or stored procedures in the public and user schemas.                                                                                                                                                                             |
| CREATE ANY TYPE      | Users can create types in the public and user schemas.                                                                                                                                                                                                                         |
| CREATE ANY SEQUENCE  | Users can create sequences in the public and user schemas.                                                                                                                                                                                                                     |

| Permission          | Description                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CREATE ANY INDEX    | Users can create indexes in the public and user schemas. The users must be granted with the permission to create tablespaces to create a partitioned table index in a tablespace.                                                                                                                                                                                                         |
| ALTER ANY TYPE      | Users have the ALTER permission on types in public and user schemas, excluding modifying the owner of a type or modifying the schema of a type.                                                                                                                                                                                                                                           |
| DROP ANY TYPE       | Users' DROP permission on types in the public and user schemas.                                                                                                                                                                                                                                                                                                                           |
| ALTER ANY SEQUENCE  | Users have the ALTER permission on sequences in public and user schemas, excluding modifying the owner of a sequence.                                                                                                                                                                                                                                                                     |
| DROP ANY SEQUENCE   | Users' DROP permission on sequences in the public and user schemas.                                                                                                                                                                                                                                                                                                                       |
| SELECT ANY SEQUENCE | Users' SELECT, USAGE, and UPDATE permissions on sequences in the public and user schemas.                                                                                                                                                                                                                                                                                                 |
| ALTER ANY INDEX     | Users' ALTER permission on indexes in the public and user schemas. To rename an index, users also need the permission to create objects in the schema where the index is located. If tablespace operations are involved, users need to have the corresponding operation permission on the tablespace. To set an index to <b>UNUSABLE</b> , users must have the DROP ANY INDEX permission. |
| DROP ANY INDEX      | Users' DROP permission on indexes in the public and user schemas.                                                                                                                                                                                                                                                                                                                         |
| CREATE ANY TRIGGER  | Users can create triggers in the public and user schemas.                                                                                                                                                                                                                                                                                                                                 |
| ALTER ANY TRIGGER   | Users' ALTER permission on triggers in the public and user schemas.                                                                                                                                                                                                                                                                                                                       |
| DROP ANY TRIGGER    | Users' DROP permission on triggers in the public and user schemas.                                                                                                                                                                                                                                                                                                                        |
| CREATE ANY SYNONYM  | Users can create synonyms in user schema.                                                                                                                                                                                                                                                                                                                                                 |
| DROP ANY SYNONYM    | Users' DROP permission on synonyms in the public and user schemas.                                                                                                                                                                                                                                                                                                                        |

 **NOTE**

If a user is granted with any ANY permission, the user has the USAGE permission on the public and user schemas but does not have the USAGE permission on the system schemas except **public** listed in [Table 13-1](#).

## Examples

### Example: Granting system permissions to a user or role

Create the **joe** user and grant the SYSADMIN permissions to it.

```
gaussdb=# CREATE USER joe PASSWORD '*****';
gaussdb=# GRANT ALL PRIVILEGES TO joe;
```

Then **joe** has the SYSADMIN permission.

### Example: Granting object permissions to a user or role

1. Revoke the SYSADMIN permission from the **joe** user. Grant the usage permission of the **tpcds** schema and all permissions on the **tpcds.reason** table to **joe**.

```
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# CREATE TABLE tpcds.reason(
 r_reason_sk INTEGER NOT NULL,
 r_reason_id CHAR(16) NOT NULL,
 r_reason_desc VARCHAR(20)
);
gaussdb=# REVOKE ALL PRIVILEGES FROM joe;
gaussdb=# GRANT USAGE ON SCHEMA tpcds TO joe;
gaussdb=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
```

Then **joe** has all permissions on the **tpcds.reason** table, including create, retrieve, update, and delete.

2. Grant the retrieve permission of **r\_reason\_sk**, **r\_reason\_id**, and **r\_reason\_desc** columns and the update permission of the **r\_reason\_desc** column in the **tpcds.reason** table to **joe**.

```
gaussdb=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON
tpcds.reason TO joe;
```

Then **joe** has the retrieve permission on the **r\_reason\_sk**, **r\_reason\_id**, and **r\_reason\_desc** columns in the **tpcds.reason** table. To enable **joe** to grant these permissions to other users, execute the following statement:

```
gaussdb=# GRANT select (r_reason_sk, r_reason_id) ON tpcds.reason TO joe WITH GRANT OPTION;
```

Grant the connection and schema creation permissions of the **testdb** database to **joe**, and allow **joe** to grant these permissions to other users.

```
gaussdb=# CREATE DATABASE testdb;
gaussdb=# GRANT create,connect on database testdb TO joe WITH GRANT OPTION;
```

Create the **tpcds\_manager** role, grant the access and object creation permissions of the **tpcds** schema to **tpcds\_manager**, but do not allow **tpcds\_manager** to grant these permissions to others.

```
gaussdb=# CREATE ROLE tpcds_manager PASSWORD '*****';
gaussdb=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
```

Grant all permissions on the **tpcds\_tbsp** tablespace to **joe**, but do not allow **joe** to grant these permissions to others.

```
gaussdb=# CREATE TABLESPACE tpcds_tbsp RELATIVE LOCATION 'tablespace/tablespace_1';
gaussdb=# GRANT ALL ON TABLESPACE tpcds_tbsp TO joe;
```

### Example: Granting the permissions of one user or role to others

1. Create the **manager** role, grant **joe**'s permissions to **manager**, and allow **manager** to grant these permissions to others.

```
gaussdb=# CREATE ROLE manager PASSWORD '*****';
gaussdb=# GRANT joe TO manager WITH ADMIN OPTION;
```

2. Create the **senior\_manager** user and grant **manager**'s permissions to it.

```
gaussdb=# CREATE ROLE senior_manager PASSWORD '*****';
gaussdb=# GRANT manager TO senior_manager;
```

### 3. Revoke permissions and delete users.

```
gaussdb=# REVOKE joe FROM manager;
gaussdb=# REVOKE manager FROM senior_manager;
gaussdb=# DROP USER manager;
gaussdb=# DROP DATABASE testdb;
```

## Example: Granting the CMK or CEK permission to other user or role

### 1. Connect to an encrypted database.

Prerequisites: You have set the parameters and used the CREATE CLIENT MASTER KEY syntax to create a master key named **MyCMK1** by referring to "Setting Encrypted Equality Queries" in *Feature Guide*.

```
gaussdb=# CREATE COLUMN ENCRYPTION KEY MyCEK1 WITH VALUES (CLIENT_MASTER_KEY =
MyCMK1, ALGORITHM = AEAD_AES_256_CBC_HMAC_SHA256);
CREATE COLUMN ENCRYPTION KEY
```

### 2. Create a role **newuser** and grant the key permission to **newuser**.

```
gaussdb=# CREATE USER newuser PASSWORD '*****';
CREATE ROLE
gaussdb=# GRANT ALL ON SCHEMA public TO newuser;
GRANT
gaussdb=# GRANT USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 to newuser;
GRANT
gaussdb=# GRANT USAGE ON CLIENT_MASTER_KEY MyCMK1 to newuser;
GRANT
```

### 3. Set the user to connect to a database and use a CEK to create an encrypted table.

```
gaussdb=# SET SESSION AUTHORIZATION newuser PASSWORD '*****';
gaussdb=> CREATE TABLE acltest1 (x int, x2 varchar(50) ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY = MyCEK1, ENCRYPTION_TYPE = DETERMINISTIC));
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'x' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=> SELECT has_cek_privilege('newuser', 'MyCEK1', 'USAGE');
has_cek_privilege

t
(1 row)
```

### 4. Revoke permissions and delete users.

```
gaussdb=# REVOKE USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 FROM newuser;
gaussdb=# REVOKE USAGE ON CLIENT_MASTER_KEY MyCMK1 FROM newuser;
gaussdb=# DROP TABLE newuser.acltest1;
gaussdb=# DROP COLUMN ENCRYPTION KEY MyCEK1;
gaussdb=# DROP CLIENT MASTER KEY MyCMK1;
gaussdb=# DROP SCHEMA IF EXISTS newuser CASCADE;
gaussdb=# REVOKE ALL ON SCHEMA public FROM newuser;
gaussdb=# DROP ROLE IF EXISTS newuser;
```

## Example: Revoking permissions and deleting roles and users

```
gaussdb=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
gaussdb=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
gaussdb=# REVOKE ALL ON TABLESPACE tpcds_tbspc FROM joe;
gaussdb=# DROP TABLESPACE tpcds_tbspc;
gaussdb=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
gaussdb=# DROP ROLE tpcds_manager;
gaussdb=# DROP ROLE senior_manager;
gaussdb=# DROP USER joe CASCADE;
gaussdb=# DROP TABLE tpcds.reason;
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Helpful Links

[REVOKE](#) and [ALTER DEFAULT PRIVILEGES](#)

## 7.14.130 IMPDP DATABASE CREATE

### Description

Specifies the preparation phase for importing a database.

### Syntax

```
IMPDP DATABASE [db_name] CREATE SOURCE = 'directory' OWNER = user [LOCAL];
```

### Parameters

- **db\_name**  
Name of the new database after the import. If not specified, the original database name is retained after the import.
- **directory**  
Data source directory of the imported database.
- **user**  
Owner of the imported database.
- **LOCAL**  
If this column is specified, data is imported to the original cluster. If this column is not specified, data is imported to a new cluster.

### Example

```
impdp database test create source = '/data1/impdp/database' owner=admin;
```

## 7.14.131 IMPDP RECOVER

### Description

Specifies the execution phase of importing a database.

### Syntax

```
IMPDP DATABASE RECOVER SOURCE = 'directory' OWNER = user [LOCAL];
```

### Parameters

- **directory**  
Data source directory of the imported database.
- **user**  
Owner of the imported database.
- **LOCAL**  
If this column is specified, data is imported to the original cluster. If this column is not specified, data is imported to a new cluster.

### Example

```
impdp database recover source = '/data1/impdp/database' owner=admin;
```

## 7.14.132 INSERT

### Description

Inserts new rows into a table.

### Precautions

- The owner of a table, users granted with the **INSERT** permission on the table, or users granted with the **INSERT ANY TABLE** permission can insert data into the table. System administrators have the permission to insert data into the table by default.
- Use of the **RETURNING** clause requires the **SELECT** permission on all columns mentioned in **RETURNING**.
- If ON DUPLICATE KEY UPDATE is used, you must have the INSERT and UPDATE permissions on the table and the SELECT permission on the columns of the UPDATE clause.
- If you use the **query** clause to insert rows from a query, you need to have the **SELECT** permission on any table or column used in the query.
- If you use the query clause to insert data from the dynamic data anonymization column, the inserted result is the anonymized value and cannot be restored.
- When you connect to a database compatible to Teradata and **td\_compatible\_truncation** is **on**, a long string will be automatically truncated. If later INSERT statements (not involving foreign tables) insert long strings to columns of CHAR- and VARCHAR-typed columns in the target table, the system will truncate the long strings to ensure no strings exceed the maximum length defined in the target table.

#### NOTE

If inserting multi-byte character data (such as Chinese characters) to a database with the character set byte encoding (SQL\_ASCII, LATIN1), and the character data crosses the truncation position, the string is truncated based on its bytes instead of characters. Unexpected result will occur in tail after the truncation. If you want correct truncation result, you are advised to adopt encoding set such as UTF8, which has no character data crossing the truncation position.

### Syntax

```
[WITH [RECURSIVE] with_query [, ...]]
INSERT [/*+ plan_hint */] INTO table_name [AS alias] ((column_name [, ...]))
 { DEFAULT VALUES
 | VALUES (({ expression | DEFAULT } [, ...]))[, ...]
 | query }
 [ON DUPLICATE KEY UPDATE { NOTHING | { column_name = { expression | DEFAULT } } [, ...] [WHERE
condition] }]
 [RETURNING { * | {output_expression [[AS] output_name] }[, ...] }];
```

### Parameters

- **WITH [ RECURSIVE ] with\_query [, ...]**  
Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table.

If **RECURSIVE** is specified, it allows a **SELECT** subquery to reference itself by name.

Format of **with\_query**:

```
with_query_name [(column_name [, ...])] AS [[NOT] MATERIALIZED]
({SELECT | VALUES | INSERT | UPDATE | DELETE})
```

– **with\_query\_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.

-- **column\_name** specifies the column name displayed in the subquery result set.

Each subquery can be a **SELECT**, **VALUES**, **INSERT**, **UPDATE** or **DELETE** statement.

- You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE.
- If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.
- If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the trunk statement to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

#### NOTE

- **INSERT ON DUPLICATE KEY UPDATE** does not support the WITH or WITH RECURSIVE clauses.
- The output of the INSERT statement displays only the number of tuples inserted into the outermost query block. For example:  
with cte as (insert into t1 values(1) returning \*) insert into t1 select \* from cte;  
Only one tuple is displayed, but two tuples are actually inserted.
- **plan\_hint** clause  
Follows the **INSERT** keyword in the */\*+ \*/* format. It is used to optimize the plan of an **INSERT** statement block. For details, see [Hint-based Tuning](#). In each statement, only the first */\*+ plan\_hint \*/* comment block takes effect as a hint. Multiple hints can be written.
- **table\_name**  
Specifies the name of the target table where data will be inserted.  
Value range: an existing table name

#### NOTE

You can use database links to perform operations on remote tables. For details, see [DATABASE LINK](#).

- **column\_name**  
Specifies the name of a column in a table.
  - The column name can be qualified with a subcolumn name or array index, if needed.

- Each column not present in the explicit or implicit column list will be filled with a default value, either its declared default value or **NULL** if there is none.
- The target column names **column\_name** can be listed in any order. If no list of column names is given at all, the default is all the columns of the table in their declared order.
- The target columns are the first *N* column names, if there are only *N* columns supplied by the VALUE clause or query.
- The values provided by the VALUE clause and query are associated with the corresponding columns from left to right in the table.

Value range: an existing column

- **expression**

Specifies an expression or a value to assign to the corresponding column.

- In the **INSERT ON DUPLICATE KEY UPDATE** statement, expression can be **VALUES(column\_name)** or **EXCLUDED.column\_name**, indicating that the value of **column\_name** corresponding to the conflict row is referenced. Note that **VALUES(column\_name)** cannot be nested in an expression (for example, **VALUES(column\_name)+1**). **EXCLUDED** is not subject to this restriction.
- If single-quotation marks are inserted in a column, the single-quotation marks need to be used for escape.
- If the expression for any column is not of the correct data type, automatic type conversion will be attempted. If the attempt fails, data insertion fails, and the system returns an error message.

- **DEFAULT**

Specifies the default value of a column. The value is **NULL** if no default value is assigned to it.

- **query**

Specifies a query statement (SELECT statement) that uses the query result as the inserted data.

- **RETURNING**

Returns the inserted rows. The syntax of the **RETURNING** list is identical to that of the output list of **SELECT**. Note that **INSERT ON DUPLICATE KEY UPDATE** does not support the **RETURNING** clause.

- **output\_expression**

Specifies an expression used to calculate the output result of the **INSERT** statement after each row is inserted.

Value range: The expression can use any column in the table. You can use the asterisk (\*) to return all columns of the inserted row.

- **output\_name**

Specifies a name to use for a returned column.

Value range: a string. It must comply with the **naming convention**.

- **ON DUPLICATE KEY UPDATE**

For a table with a unique constraint (**UNIQUE INDEX** or **PRIMARY KEY**), if the inserted data violates the unique constraint, the **UPDATE** clause is executed to update the conflicting rows. If the clause of **UPDATE** is **NOTHING**, no operation will be performed.

For a table without a unique constraint, only insert is performed.

- Triggers are supported. The execution sequence of triggers is determined by the actual execution process.
  - Run the **INSERT** command to trigger the **before insert** and **after insert** triggers.
  - Executing UPDATE will trigger the BEFORE INSERT, BEFORE UPDATE, and AFTER UPDATE.
  - Executing UPDATE NOTHING will trigger the BEFORE INSERT.
- The unique constraint or primary key of **DEFERRABLE** is not supported.
- If a table has multiple unique constraints and the inserted data violates multiple unique constraints, only the first row that has a conflict is updated. (The check sequence is closely related to index maintenance. Generally, the conflict check is performed on the index that is created first.)
- Distribution columns and unique index columns cannot be updated.
- The **WHERE** clause of **UPDATE** does not contain sublinks.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
);

-- Insert a record into a table:
gaussdb=# INSERT INTO tpcds.reason(r_reason_sk, r_reason_id, r_reason_desc) VALUES (0,
'AAAAAAAAAAAAAAAAAAAA', 'reason0');

-- Create the tpcds.reason_t2 table:
gaussdb=# CREATE TABLE tpcds.reason_t2
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
);

-- Insert a record into a table:
gaussdb=# INSERT INTO tpcds.reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1,
'AAAAAAAABAAAAAAA', 'reason1');

-- Insert a record into the table, which is equivalent to the previous syntax:
gaussdb=# INSERT INTO tpcds.reason_t2 VALUES (2, 'AAAAAAAABAAAAAAA', 'reason2');

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.reason_t2 VALUES (3, 'AAAAAAAACAAAAAAA', 'reason3'),(4,
'AAAAAAAADAAAAAAA', 'reason4'),(5, 'AAAAAAAEEAAAAAAA', 'reason5');

-- Insert records whose r_reason_sk in the tpcds.reason table is less than 5:
gaussdb=# INSERT INTO tpcds.reason_t2 SELECT * FROM tpcds.reason WHERE r_reason_sk <5;

-- Create a unique index for the table:
gaussdb=# CREATE UNIQUE INDEX reason_t2_u_index ON tpcds.reason_t2(r_reason_sk);

-- Insert multiple records into the table. If the records conflict, update the r_reason_id column in the
```

```
conflict data row to BBBBBBBCCAAAAAAA.
gaussdb=# INSERT INTO tpcds.reason_t2 VALUES (5, 'BBBBBBBCCA', 'reason5'), (6,
'AAAAAADAAAA', 'reason6') ON DUPLICATE KEY UPDATE r_reason_id = 'BBBBBBBCCA';

-- Delete the tpcds.reason_t2 table.
gaussdb=# DROP TABLE tpcds.reason_t2;

-- Delete the tpcds.reason table.
gaussdb=# DROP TABLE tpcds.reason;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Suggestions

- **VALUES**

When you run the INSERT statement to insert data in batches, you are advised to combine multiple records into one statement to improve data loading performance. Example: **INSERT INTO sections VALUES (30, 'Administration', 31, 1900), (40, 'Development', 35, 2000), (50, 'Development', 60, 2001);**

If values of an INSERT statement are distributed on a DN, GaussDB can push the statement down to the corresponding DN for execution. Currently, only constants, simple expressions, and pushdown functions (**provolatile** in **pg\_proc** is set to 'i') are supported. If a column in the table has a default value, the value must be a constant or a simple expression. Neither a single-value statement nor a multi-value statement can be pushed down to a single DN.

## 7.14.133 IMPDP TABLE

### Description

Specifies the execution phase of importing a table.

### Syntax

```
IMPDP TABLE [AS table_name] SOURCE = 'directory' OWNER = user;
```

### Parameters

- **table\_name**  
Name of the target table after the import. If not specified, the original table name is retained after the import.
- **directory**  
Data source directory of the imported table.
- **user**  
Owner of the imported table.

### Example

```
impdp table source = '/data1/impdp/table0' owner=admin;
```

## 7.14.134 IMPDP TABLE PREPARE

### Description

Specifies the preparation phase for importing a table.

### Syntax

```
IMPDP TABLE PREPARE SOURCE = 'directory' OWNER = user;
```

### Parameters

- **directory**  
Data source directory of the imported table.
- **user**  
Owner of the imported table.

### Example

```
impdp table prepare source = '/data1/impdp/table0' owner=admin;
```

## 7.14.135 LOCK

### Description

LOCK TABLE obtains a table-level lock.

GaussDB always tries to select the lock mode with minimum constraints when automatically requesting a lock for a statement referenced by a table. Use LOCK if users need a more strict lock mode. For example, suppose an application runs a transaction at the Read Committed isolation level and needs to ensure that data in a table remains stable in the duration of the transaction. To achieve this, you could obtain SHARE lock mode over the table before the query. This will prevent concurrent data changes and ensure subsequent reads of the table see a stable view of committed data. It is because the **SHARE** lock mode conflicts with the **ROW EXCLUSIVE** lock acquired by writers, and your **LOCK TABLE name IN SHARE MODE** statement will wait until any concurrent holders of **ROW EXCLUSIVE** mode locks commit or roll back. Therefore, once you obtain the lock, there are no uncommitted writes outstanding; furthermore none can begin until you release the lock.

### Precautions

- **LOCK TABLE** is useless outside a transaction block: the lock would remain held only to the completion of the statement. If **LOCK TABLE** is out of any transaction block, an error is reported.
- If no lock mode is specified, then **ACCESS EXCLUSIVE**, the most restrictive mode, is used.
- **LOCK TABLE ... IN ACCESS SHARE MODE** requires SELECT privileges on the target table. All other forms of LOCK require the UPDATE or DELETE permission.
- There is no **UNLOCK TABLE** statement. Locks are always released at transaction end.

- **LOCK TABLE** only deals with table-level locks, and so the mode names involving **ROW** are all misnomers. These mode names should generally be read as indicating the intention of the user to acquire row-level locks within the locked table. Also, **ROW EXCLUSIVE** mode is a shareable table lock. Note that all lock modes have the same semantics as long as **LOCK TABLE** is involved. The only difference lies in whether locks conflict with each other. For details about the rules, see [Table 7-167](#).
- If the `xc_maintenance_mode` parameter is not enabled, an error is reported when an **ACCESS EXCLUSIVE** lock is applied for a system catalog.
- Only the redistribution tool can use the automatic **CANCEL** service interface.

## Syntax

```
LOCK [TABLE] {[ONLY] name [, ...]} {name [*]} [, ...]
 [IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE
ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE]
 [NOWAIT][CANCELABLE];
```

## Parameters

**Table 7-167** Lock mode conflicts

| Requested Lock Mode/<br>Current Lock Mode | ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE |
|-------------------------------------------|--------------|-----------|---------------|------------------------|-------|---------------------|-----------|------------------|
| ACCESS SHARE                              | -            | -         | -             | -                      | -     | -                   | -         | X                |
| ROW SHARE                                 | -            | -         | -             | -                      | -     | -                   | X         | X                |
| ROW EXCLUSIVE                             | -            | -         | -             | -                      | X     | X                   | X         | X                |
| SHARE UPDATE EXCLUSIVE                    | -            | -         | -             | X                      | X     | X                   | X         | X                |
| SHARE                                     | -            | -         | X             | X                      | -     | X                   | X         | X                |
| SHARE ROW EXCLUSIVE                       | -            | -         | X             | X                      | X     | X                   | X         | X                |

| Requested Lock Mode/<br>Current Lock Mode | ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE |
|-------------------------------------------|--------------|-----------|---------------|------------------------|-------|---------------------|-----------|------------------|
| EXCLUSIVE                                 | -            | X         | X             | X                      | X     | X                   | X         | X                |
| ACCESS EXCLUSIVE                          | X            | X         | X             | X                      | X     | X                   | X         | X                |

**LOCK** parameters are as follows:

- **name**

Specifies the name (optionally schema-qualified) of an existing table to lock. Tables are locked one-by-one in the order specified in the **LOCK TABLE** statement.

Value range: an existing table name

 **NOTE**

You can use database links to perform operations on remote tables. For details, see [DATABASE LINK](#).

- **ONLY**

If **ONLY** is specified, only that table is locked. If it is not specified, the table and all its sub-tables are locked.

- **ACCESS SHARE**

Allows only read operations on a table. In general, any SQL statements that only read a table and do not modify it will acquire this lock mode. The **SELECT** statement acquires a lock of this mode on referenced tables.

- **ROW SHARE**

Allows concurrent read of a table but does not allow any other operations on the table.

**SELECT FOR UPDATE** and **SELECT FOR SHARE** automatically acquire the **ROW SHARE** lock on the target table and add the **ACCESS SHARE** lock to other referenced tables except **FOR SHARE** and **FOR UPDATE**.

For a partitioned table, **SELECT FOR SHARE** obtains the **ROW EXCLUSIVE** lock of the partition object on the DN for concurrency control.

- **ROW EXCLUSIVE**

Allows concurrent read of a table but does not allow modification of data in the table like **ROW SHARE**. **UPDATE**, **DELETE**, and **INSERT** automatically acquire this lock on the target table and add the **ACCESS SHARE** lock to other

referenced tables. Generally, all statements that modify table data acquire the **ROW EXCLUSIVE** lock for tables.

- **SHARE UPDATE EXCLUSIVE**

Protects a table against concurrent schema changes and **VACUUM** runs.

The **VACUUM** (without **FULL**), **ANALYZE**, and **CREATE INDEX CONCURRENTLY** statements automatically request this lock.

- **SHARE**

Allows concurrent queries of a table but does not allow modification of the table.

The **CREATE INDEX** (without **CONCURRENTLY**) statement automatically requests this lock.

- **SHARE ROW EXCLUSIVE**

Protects a table against concurrent data changes, and is self-exclusive so that only one session can hold it at a time.

No SQL statements automatically acquire this lock mode.

- **EXCLUSIVE**

Allows concurrent queries of the target table but does not allow any other operations.

This mode allows only concurrent **ACCESS SHARE** locks; that is, only reads from the table can proceed in parallel with a transaction holding this lock mode.

No SQL statements automatically acquire this lock mode on user tables. However, it will be acquired on some system catalogs in case of some operations.

- **ACCESS EXCLUSIVE**

Guarantees that the holder is the only transaction accessing the table in any way.

Acquired by the **ALTER TABLE**, **DROP TABLE**, **TRUNCATE**, **REINDEX**, **CLUSTER**, and **VACUUM FULL** statements.

This is also the default lock mode for **LOCK TABLE** statements that do not specify a mode explicitly.

- **NOWAIT**

Specifies that **LOCK TABLE** does not wait for any conflicting locks to be released. If the lock cannot be obtained immediately, the command exits and an error message is displayed.

If a table-level lock is obtained without specifying **NOWAIT** and other mutex locks exist, the system waits for other locks to be released.

- **CANCELABLE**

Allows the waiting thread to send **CANCEL** signals to the holding threads and waiting threads.

Only the redistribution tool can use this parameter. An error message is displayed when the parameter is used by users.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;
```

```
-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason(
r_reason_sk INTEGER NOT NULL,
r_reason_id CHAR(16) NOT NULL,
r_reason_desc INTEGER
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAAABAAAAAAA', '18'),(5, 'AAAAAAAACAAAAAAA',
'362'),(7, 'AAAAAAAADAAAAAAA', '585');

-- Obtain a SHARE ROW EXCLUSIVE lock on a primary key table when going to perform a delete operation.
gaussdb=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

gaussdb=# START TRANSACTION;

gaussdb=# LOCK TABLE tpcds.reason_t1 IN SHARE ROW EXCLUSIVE MODE;

gaussdb=# DELETE FROM tpcds.reason_t1 WHERE r_reason_desc IN(SELECT r_reason_desc FROM
tpcds.reason_t1 WHERE r_reason_sk < 6);

gaussdb=# DELETE FROM tpcds.reason_t1 WHERE r_reason_sk = 7;

gaussdb=# COMMIT;

-- Delete the tpcds.reason_t1 table.
gaussdb=# DROP TABLE tpcds.reason_t1;

-- Delete the table.
gaussdb=# DROP TABLE tpcds.reason;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## 7.14.136 MOVE

### Description

Repositions a cursor without retrieving any data. MOVE works exactly like the [FETCH](#) statement, except it only repositions the cursor and does not return rows.

### Syntax

```
MOVE [direction [FROM | IN]] cursor_name;
```

The **direction** clause specifies optional parameters.

```
NEXT
| PRIOR
| FIRST
| LAST
| ABSOLUTE count
| RELATIVE count
| count
| ALL
| FORWARD
| FORWARD count
| FORWARD ALL
| BACKWARD
| BACKWARD count
| BACKWARD ALL
```

## Parameters

The MOVE statement parameters are the same as FETCH statement parameters. For details, see [Parameters](#) in section "FETCH."

### NOTE

On successful completion, the MOVE statement returns a "**MOVE count**" tag, where *count* indicates the number of rows that the FETCH statement with the same parameters would have returned (possibly zero).

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason(
r_reason_sk INTEGER NOT NULL,
r_reason_id CHAR(16) NOT NULL,
r_reason_desc VARCHAR(40)
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAABAAAAAAA', 'XXXXXXXX'),(2,
'AAAAAAAACAAAAAAA', 'XXXXXXXX'),(3, 'AAAAAAAADAAAAAAA', 'XXXXXXXX'),(4, 'AAAAAAAEEAAAAAAA',
'Not the product that was ordered'),(5, 'AAAAAAAFAAAAAAAA', 'Parts missing'),(6, 'AAAAAAAAGAAAAAAA',
'Does not work with a product that I have'),(7, 'AAAAAAAHAAAAAAA', 'Gift exchange');

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Define a cursor named cursor1.
gaussdb=# CURSOR cursor1 FOR SELECT * FROM tpcds.reason;

-- Skip the first three rows of cursor1:
gaussdb=# MOVE FORWARD 3 FROM cursor1;

-- Fetch the first four rows from cursor1:
gaussdb=# FETCH 4 FROM cursor1;
 r_reason_sk | r_reason_id | r_reason_desc
-----+-----
4 | AAAAAAAEEAAAAAAA | Not the product that was
ordered
5 | AAAAAAAFAAAAAAAA | Parts missing
6 | AAAAAAAGAAAAAAA | Does not work with a product that I
have
7 | AAAAAAAHAAAAAAA | Gift
exchange
(4 rows)

-- Close the cursor.
gaussdb=# CLOSE cursor1;

-- End the transaction.
gaussdb=# END;

-- Delete the table.
gaussdb=# DROP TABLE tpcds.reason;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Helpful Links

[CLOSE](#) and [FETCH](#)

## 7.14.137 MERGE INTO

### Description

MERGE INTO conditionally matches data in a target table with that in a source table. If data matches, UPDATE is executed on the target table; if data does not match, INSERT is executed. You can use this syntax to run **UPDATE** and **INSERT** at a time for convenience

### Precautions

- You have the **INSERT** and **UPDATE** permissions for the target table and the **SELECT** permission for the source table.
- **MERGE INTO** cannot be executed during redistribution.
- If the source table of the **MERGE INTO** operation contains data columns that are dynamically anonymized, the result of inserting data to or updating data in the target table is the anonymized value and cannot be restored.

### Syntax

```
MERGE [/*+ plan_hint */] INTO table_name [[AS] alias]
USING { { table_name | view_name } | subquery } [[AS] alias]
ON (condition)
[
 WHEN MATCHED THEN
 UPDATE SET { column_name = { expression | subquery | DEFAULT } |
 (column_name [, ...]) = ({ expression | subquery | DEFAULT } [, ...]) } [, ...]
 [WHERE condition]
]
[
 WHEN NOT MATCHED THEN
 INSERT { DEFAULT VALUES |
 ((column_name [, ...])) VALUES ({ expression | subquery | DEFAULT } [, ...]) [, ...] [WHERE condition] }
];
NOTICE: 'subquery' in the UPDATE and INSERT clauses are only available in CENTRALIZED mode!
```

### Parameters

- **plan\_hint** clause  
Follows the MERGE keyword in the */\*+ \*/* format. It is used to optimize the plan of a MERGE statement block. For details, see [Hint-based Tuning](#). In each statement, only the first */\*+ plan\_hint \*/* comment block takes effect as a hint. Multiple hints can be written.
- **INTO** clause  
Specifies the target table that is being updated or has data being inserted. If the target table is a replication table, the default value of a column (such as auto-increment column) in the target table cannot be the volatile function. If **enable\_stream\_operator** is set to **off**, the target table must contain a primary key or UNIQUE and NOT NULL constraints.
  - **table\_name**  
Specifies the name of the target table.
  - **alias**  
Specifies the alias of the target table.  
Value range: a string. It must comply with the [naming convention](#).

- **USING clause**  
Specifies the source table, which can be a table, view, or subquery. If the target table is a replication table, the **USING** clause cannot contain non-replication tables.
- **ON clause**  
Specifies the condition used to match data between the source and target tables. Columns in the condition cannot be updated.
- **WHEN MATCHED clause**  
Performs **UPDATE** if data in the source table matches that in the target table based on the condition.  
Distribution keys cannot be updated. System catalogs and system columns cannot be updated.
- **WHEN NOT MATCHED clause**  
Performs **INSERT** if data in the source table does not match that in the target table based on the condition.  
An **INSERT** clause cannot contain multiple **VALUES**.  
The order of **WHEN MATCHED** and **WHEN NOT MATCHED** clauses can be reversed. One of them can be used by default, but they cannot be both used at one time. Two **WHEN MATCHED** or **WHEN NOT MATCHED** clauses cannot be specified at the same time.
- **DEFAULT**  
Specifies the default value of a column.  
The value is **NULL** if no default value is assigned to it.
- **WHERE condition**  
Specifies the conditions for the **UPDATE** and **INSERT** clauses. The two clauses will be executed only when the conditions are met. The default value can be used. System columns cannot be referenced in **WHERE** condition. You are advised not to use numeric types such as **int** as conditions, because such types can be implicitly converted to **bool** values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

## Examples

```
-- Create the target table products and source table newproducts, and insert data to them.
gaussdb=# CREATE TABLE products
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

gaussdb=# INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrncs');
gaussdb=# INSERT INTO products VALUES (1502, 'olympus is50', 'electrncs');
gaussdb=# INSERT INTO products VALUES (1600, 'play gym', 'toys');
gaussdb=# INSERT INTO products VALUES (1601, 'lamaze', 'toys');
gaussdb=# INSERT INTO products VALUES (1666, 'harry potter', 'dvd');

gaussdb=# CREATE TABLE newproducts
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
```

```
);

gaussdb=# INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrnrcs');
gaussdb=# INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
gaussdb=# INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
gaussdb=# INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');

-- Run MERGE INTO.
gaussdb=# MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
 UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE p.product_name !=
'play gym'
WHEN NOT MATCHED THEN
 INSERT VALUES (np.product_id, np.product_name, np.category) WHERE np.category = 'books';
MERGE 4

-- Query updates.
gaussdb=# SELECT * FROM products ORDER BY product_id;
 product_id | product_name | category
-----+-----+-----
 1501 | vivitar 35mm | electrncs
 1502 | olympus camera | electrncs
 1600 | play gym | toys
 1601 | lamaze | toys
 1666 | harry potter | toys
 1700 | wait interface | books
(6 rows)

-- Drop the table.
gaussdb=# DROP TABLE products;
gaussdb=# DROP TABLE newproducts;
```

## 7.14.138 PREDICT BY

This syntax is not supported in distributed scenarios.

## 7.14.139 PREPARE

### Description

Creates a prepared statement.

A prepared statement is a performance optimizing object on the server. When the PREPARE statement is executed, the specified query is parsed, analyzed, and rewritten. When EXECUTE is executed, the prepared statement is planned and executed. This avoids repetitive parsing and analysis. After the PREPARE statement is created, it exists throughout the database session. Once it is created (even if in a transaction block), it will not be deleted when a transaction is rolled back. It can only be deleted by explicitly calling **DEALLOCATE** or automatically deleted when the session ends.

### Syntax

```
PREPARE name [(data_type [, ...])] AS statement;
```

### Parameters

- **name**  
Specifies the name of a prepared statement. It must be unique in the session.

- **data\_type**  
Specifies the type of an argument.
- **statement**  
Specifies a SELECT, INSERT, UPDATE, DELETE, MERGE INTO, or VALUES statement.

## Examples

See [Examples](#) in section "EXECUTE."

## Helpful Links

[DEALLOCATE](#)

# 7.14.140 PREPARE TRANSACTION

## Description

Prepares the current transaction for two-phase commit.

After this statement, the transaction is no longer associated with the current session; instead, its state is fully stored on disk, and there is a high probability that it can be committed successfully, even if a database crash occurs before the commit is requested.

Once prepared, a transaction can later be committed or rolled back with [COMMIT PREPARED](#) or [ROLLBACK PREPARED](#), respectively. Those statements can be issued from any session, not only the one that executed the original transaction.

From the point of view of the issuing session, **PREPARE TRANSACTION** is not unlike a **ROLLBACK** statement: after executing it, there is no active current transaction, and the effects of the prepared transaction are no longer visible. (The effects will become visible again if the transaction is committed.)

If the **PREPARE TRANSACTION** statement fails for any reason, it becomes a **ROLLBACK** and the current transaction is canceled.

## Precautions

- The transaction function is maintained automatically by the database, and should be not visible to users.
- The distributed system does not allow users to call the customized PREPARE TRANSACTION operation.
- When running the **PREPARE TRANSACTION** statement, increase the value of **max\_prepared\_transactions** in configuration file **postgresql.conf**. You are advised to set it to a value not less than that of **max\_connections** so that one pending prepared transaction is available for each session.

## Syntax

```
PREPARE TRANSACTION transaction_id;
```

## Parameters

- **transaction\_id**  
Specifies an arbitrary identifier that later identifies this transaction for **COMMIT PREPARED** or **ROLLBACK PREPARED**. The identifier must be different from those for current prepared transactions.  
Value range: The identifier must be written as a string literal, and must be less than 200 bytes long.

## Helpful Links

[COMMIT PREPARED](#) and [ROLLBACK PREPARED](#)

## 7.14.141 PURGE

### Description

The PURGE statement can be used to:

- Clear tables or indexes from the recycle bin and release all space related to the objects.
- Clear the recycle bin.
- Clear the objects of a specified tablespace in the recycle bin.

### Precautions

- The PURGE operation supports tables (**PURGE TABLE**), indexes (**PURGE INDEX**), and recycle bins (**PURGE RECYCLEBIN**).
- The permission requirements for performing the PURGE operation are as follows:
  - **PURGE TABLE**: The user must be the owner of the table and must have the USAGE permission on the schema to which the table belongs. System administrators have this permission by default.
  - **PURGE INDEX**: The user must be the owner of the index and have the USAGE permission on the schema to which the index belongs. System administrators have this permission by default.
  - **PURGE RECYCLEBIN**: Common users can clear only the objects owned by themselves in the recycle bin. In addition, the user must have the USAGE permission of the schema to which the objects belong. By default, system administrators can clear all objects in the recycle bin.

### Prerequisites

- The **enable\_recyclebin** parameter has been enabled to enable the recycle bin. Contact an administrator for details about how to use the parameter.
- The **recyclebin\_retention\_time** parameter has been set for specifying the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires. Contact an administrator for details about how to use the parameter.

## Syntax

```
PURGE { TABLE [schema_name.]table_name
 | INDEX index_name
 | RECYCLEBIN
 }
```

## Parameters

- **schema\_name**  
Schema name.
- **TABLE [ schema\_name. ] table\_name**  
Clears a specified table in the recycle bin.
- **INDEX index\_name**  
Clears a specified index in the recycle bin.
- **RECYCLEBIN**  
Clears all objects in the recycle bin.

## Examples

```
-- Create the tpcds role.
gaussdb=# CREATE ROLE tpcds IDENTIFIED BY '*****';

-- Create the reason_table_space tablespace.
gaussdb=# CREATE TABLESPACE REASON_TABLE_SPACE1 owner tpcds RELATIVE location 'tablespace/
tsp_reason1';

-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason_t1 table in the tablespace.
gaussdb=# CREATE TABLE tpcds.reason_t1
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
) with(storage_type=ustore) tablespace reason_table_space1;
-- Create the tpcds.reason_t2 table in the tablespace.
gaussdb=# CREATE TABLE tpcds.reason_t2
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
) with(storage_type=ustore) tablespace reason_table_space1;
-- Create the tpcds.reason_t3 table in the tablespace.
gaussdb=# CREATE TABLE tpcds.reason_t3
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
) with(storage_type=ustore) tablespace reason_table_space1;
-- Create an index on the tpcds.reason_t1 table.
gaussdb=# CREATE INDEX index_t1 on tpcds.reason_t1(r_reason_id);
gaussdb=# DROP TABLE tpcds.reason_t1;
gaussdb=# DROP TABLE tpcds.reason_t2;
gaussdb=# DROP TABLE tpcds.reason_t3;
-- View the recycle bin.
gaussdb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
 rcyname | rcyoriginname | rcytablespace
-----+-----+-----
BIN$16409$2CEE988==$0 | reason_t1 | 16408
BIN$16412$2CF2188==$0 | reason_t2 | 16408
BIN$16415$2CF2EC8==$0 | reason_t3 | 16408
```

```
BIN$16418$2CF3EC8==$0 | index_t1 | 0
(4 rows)
-- Purge the table.
gaussdb=# PURGE TABLE tpcds.reason_t3;
gaussdb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
 rcyname | rcyoriginname | rcytablespace
-----+-----+-----
BIN$16409$2CEE988==$0 | reason_t1 | 16408
BIN$16412$2CF2188==$0 | reason_t2 | 16408
BIN$16418$2CF3EC8==$0 | index_t1 | 0
(3 rows)
-- Purge the index.
gaussdb=# PURGE INDEX tpcds.index_t1;
gaussdb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
 rcyname | rcyoriginname | rcytablespace
-----+-----+-----
BIN$16409$2CEE988==$0 | reason_t1 | 16408
BIN$16412$2CF2188==$0 | reason_t2 | 16408
(2 rows)
-- Purge all objects in the recycle bin.
gaussdb=# PURGE recyclebin;
gaussdb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
 rcyname | rcyoriginname | rcytablespace
-----+-----+-----
(0 rows)

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## 7.14.142 REASSIGN OWNED

### Description

Changes the owner of the database object.

REASSIGN OWNED changes the database object owner of an old role to a new role.

### Precautions

- REASSIGN OWNED is often executed before role deletion.
- To run **REASSIGN OWNED**, you must have the permissions of the original and target roles.

### Syntax

```
REASSIGN OWNED BY old_role [, ...] TO new_role;
```

### Parameters

- **old\_role**  
Specifies the role name of the old owner.
- **new\_role**  
Specifies the role name of the new owner. Note: Only the initial user can use the REASSIGN OWNED syntax to change the owner to the initial user.

### Examples

None

## 7.14.143 REINDEX

### Description

Rebuilds an index using the data stored in the index's table, replacing the old copy of the index.

There are several scenarios in which **REINDEX** can be used:

- An index has become corrupted, and no longer contains valid data.
- An index has become "bloated", that is, it contains many empty or nearly-empty pages.
- You have altered a storage parameter (such as a fill factor) for an index, and wish that the change takes full effect.

### Precautions

- **REINDEX DATABASE** and **REINDEX SYSTEM** type cannot be performed in transaction blocks.
- When **REINDEX CONCURRENTLY** is used to rebuild indexes online, the index sequence in the table changes, and the query plan may change.

### Syntax

- Rebuild a general index.  
`REINDEX { INDEX | TABLE | DATABASE | SYSTEM } [CONCURRENTLY] name [ FORCE ];`
- Rebuild an index partition.  
`REINDEX { INDEX| TABLE} name  
PARTITION partition_name [ FORCE ];`

### Parameters

- **INDEX**  
Rebuilds the specified index.
- **TABLE**  
Rebuilds all indexes of a specified table. The TOAST table (if any) of the table is reindexed as well. If an index on the table has been invalidated by running **alter unusable**, the index cannot be rebuilt. Indexes in the TOAST table cannot be rebuilt when specifying the **CONCURRENTLY** option.
- **DATABASE**  
Rebuilds all indexes within the current database. Indexes in the TOAST table within the current database cannot be rebuilt when specifying the **CONCURRENTLY** option.
- **SYSTEM**  
Rebuilds all indexes on system catalogs within the current database. Indexes on user tables are not processed.
- **CONCURRENTLY**  
Rebuilds an index (with ShareUpdateExclusiveLock) in non-blocking DML mode. When an index is rebuilt, other statements cannot access the table on which the index depends. If this keyword is specified, DML is not blocked during the recreation. Indexes in system catalogs cannot be rebuilt online.

REINDEX INTERNAL TABLE CONCURRENTLY and REINDEX SYSTEM CONCURRENTLY are not supported. When REINDEX DATABASE CONCURRENTLY is executed, all indexes on user tables in the current database are rebuilt online (indexes on system catalogs are not processed). REINDEX CONCURRENTLY cannot be executed within a transaction. Only B-tree and UB-tree indexes can be created online and only common, global, and local indexes are supported. Online concurrent index rebuilding supports only Astore ordinary indexes, global indexes, and local indexes. Ustore indexes are not supported. If online index recreating fails, invalid new indexes may be left. If the system cannot automatically clear the invalid indexes (for example, the database is shut down), you need to manually clear the invalid indexes (using the DROP INDEX statement) as soon as possible to prevent more resources from being occupied. Generally, the extension of an invalid index name is **\_ccnew**. The execution of REINDEX INDEX CONCURRENTLY adds a four-level session lock to the table and its first several phases are similar to those of CREATE INDEX CONCURRENTLY. Therefore, the execution may be suspended or deadlocked, which is similar to that of CREATE INDEX CONCURRENTLY. For example, if two sessions perform the REINDEX CONCURRENTLY operation on the same index or table at the same time, a deadlock occurs. For details, see [CONCURRENTLY](#).

- **name**

Specifies the name of the index, table, or database whose index needs to be rebuilt. Tables and indexes can be schema-qualified.

 **NOTE**

REINDEX DATABASE and REINDEX SYSTEM can create indexes for only the current database. Therefore, **name** must be the same as the current database name.

- **FORCE**

Discarded parameter. It is currently reserved for compatibility with earlier versions.

- **partition\_name**

Specifies the name of the partition or index partition to be rebuilt.

Value range:

- If REINDEX INDEX is used, specify the name of an index partition.
- If REINDEX TABLE is used, specify the name of a partition.

---

**NOTICE**

REINDEX DATABASE and REINDEX SYSTEM cannot be performed in transaction blocks.

---

 **CAUTION**

REINDEX and REINDEX CONCURRENTLY do not support separate operations on TOAST tables or TOAST indexes.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer table.
gaussdb=# CREATE TABLE tpcds.customer(
c_customer_sk INTEGER NOT NULL,
c_customer_id CHAR(16) NOT NULL
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAAABAAAAAAA'),(5, 'AAAAAAAACAAAAAAA'),
(10, 'AAAAAAAADAAAAAAA');

-- Create a row-store table tpcds.customer_t1 and create an index on the c_customer_sk column in the
table.
gaussdb=# CREATE TABLE tpcds.customer_t1
(
c_customer_sk integer not null,
c_customer_id char(16) not null,
c_current_demo_sk integer ,
c_current_hdemo_sk integer ,
c_current_addr_sk integer ,
c_first_shipto_date_sk integer ,
c_first_sales_date_sk integer ,
c_salutation char(10) ,
c_first_name char(20) ,
c_last_name char(30) ,
c_preferred_cust_flag char(1) ,
c_birth_day integer ,
c_birth_month integer ,
c_birth_year integer ,
c_birth_country varchar(20) ,
c_login char(13) ,
c_email_address char(50) ,
c_last_review_date char(10)
)
WITH (orientation = row);

gaussdb=# CREATE INDEX tpcds_customer_index1 ON tpcds.customer_t1 (c_customer_sk);

gaussdb=# INSERT INTO tpcds.customer_t1 SELECT * FROM tpcds.customer WHERE c_customer_sk < 10;

-- Rebuild a single index.
gaussdb=# REINDEX INDEX tpcds.tpcds_customer_index1;

-- Rebuild a single index online.
gaussdb=# REINDEX INDEX CONCURRENTLY tpcds.tpcds_customer_index1;

-- Rebuild all indexes in the tpcds.customer_t1 table:
gaussdb=# REINDEX TABLE tpcds.customer_t1;

-- Rebuild all indexes in the tpcds.customer_t1 table online.
gaussdb=# REINDEX TABLE CONCURRENTLY tpcds.customer_t1;

-- Delete the tpcds.customer_t1 table:
gaussdb=# DROP TABLE tpcds.customer_t1;

-- Delete the tpcds.customer table.
gaussdb=# DROP TABLE tpcds.customer;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Suggestions

- DATABASE

You are advised not to reindex a database in a transaction.

- SYSTEM

You are advised not to reindex a system catalog in a transaction.

## 7.14.144 REFRESH INCREMENTAL MATERIALIZED VIEW

### Function

**REFRESH INCREMENTAL MATERIALIZED VIEW** refreshes a materialized view in materialized mode.

### Precautions

- Incremental refresh supports only incremental materialized views.
- To refresh a materialized view, you must have the SELECT permission on the base table.

### Syntax

```
REFRESH INCREMENTAL MATERIALIZED VIEW mv_name;
```

### Parameter Description

- **mv\_name**  
Name of the materialized view to be refreshed.

### Examples

```
-- Create an ordinary table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int);

-- Create a fast-refresh materialized view.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

-- Write data to the base table.
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Fast refresh the fast-refresh materialized view my_imv.
gaussdb=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

-- Delete the fast-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_imv;

-- Delete the my_table table.
gaussdb=# DROP TABLE my_table;
```

### Helpful Links

[ALTER MATERIALIZED VIEW](#), [CREATE INCREMENTAL MATERIALIZED VIEW](#), [CREATE MATERIALIZED VIEW](#), [CREATE TABLE](#), [DROP MATERIALIZED VIEW](#), and [REFRESH MATERIALIZED VIEW](#)

## 7.14.145 REFRESH MATERIALIZED VIEW

### Function

**REFRESH MATERIALIZED VIEW** refreshes materialized views in full refresh mode.

## Precautions

- Full refreshing can be performed on both full and incremental materialized views.
- To refresh a materialized view, you must have the SELECT permission on the base table.

## Syntax

```
REFRESH MATERIALIZED VIEW mv_name;
```

## Parameter Description

- **mv\_name**  
Name of the materialized view to be refreshed.

## Examples

```
-- Create an ordinary table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int);

-- Create a complete-refresh materialized view.
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

-- Create a fast-refresh materialized view.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

-- Write data to the base table.
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Completely refresh the complete-refresh materialized view my_mv.
gaussdb=# REFRESH MATERIALIZED VIEW my_mv;

-- Completely refresh the fast-refresh materialized view my_imv.
gaussdb=# REFRESH MATERIALIZED VIEW my_imv;

-- Delete a fast-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_imv;

-- Delete a complete-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_mv;

-- Delete the my_table table.
gaussdb=# DROP TABLE my_table;
```

## Helpful Links

[ALTER MATERIALIZED VIEW](#), [CREATE INCREMENTAL MATERIALIZED VIEW](#), [CREATE MATERIALIZED VIEW](#), [CREATE TABLE](#), [DROP MATERIALIZED VIEW](#), and [REFRESH INCREMENTAL MATERIALIZED VIEW](#)

## 7.14.146 RELEASE SAVEPOINT

### Description

Destroys a savepoint previously defined in the current transaction.

Destroying a savepoint makes it unavailable as a rollback point, but it has no other user visible behavior. It does not undo the effects of statements executed after the savepoint was established. To do that, use ROLLBACK TO SAVEPOINT.

Destroying a savepoint when it is no longer needed allows the system to reclaim some resources earlier than transaction end.

RELEASE SAVEPOINT also destroys all savepoints that were established after the named savepoint was established.

## Precautions

- Specifying a savepoint name that was not previously defined causes an error.
- It is not possible to release a savepoint when the transaction is in an aborted state.
- If multiple savepoints have the same name, only the one that was most recently defined is released.

## Syntax

```
RELEASE [SAVEPOINT] savepoint_name;
```

## Parameters

### **savepoint\_name**

Specifies the name of the savepoint you want to destroy.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create a table.
gaussdb=# CREATE TABLE tpcds.table1(a int);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO tpcds.table1 VALUES (3);

-- Establish a savepoint.
gaussdb=# SAVEPOINT my_savepoint;

-- Insert data.
gaussdb=# INSERT INTO tpcds.table1 VALUES (4);

-- Delete the savepoint.
gaussdb=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
gaussdb=# COMMIT;

-- Query the table content, which should contain both 3 and 4.
gaussdb=# SELECT * FROM tpcds.table1;

-- Delete the table.
gaussdb=# DROP TABLE tpcds.table1;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Helpful Links

[SAVEPOINT](#) and [ROLLBACK TO SAVEPOINT](#)

## 7.14.147 RESET

### Description

Restores GUC parameters to their default values. The default values are parameter default values compiled in the **postgresql.conf** configuration file.

**RESET** is an alternative spelling for:

```
SET configuration_parameter TO DEFAULT;
```

### Precautions

RESET and SET have the same transaction behavior. Their impact will be rolled back.

### Syntax

```
RESET {configuration_parameter | CURRENT_SCHEMA | TIME_ZONE | TRANSACTION ISOLATION LEVEL |
SESSION AUTHORIZATION | ALL };
```

### Parameters

- **configuration\_parameter**  
Specifies the name of a settable GUC parameter.  
Value range: GUC parameters. You can view them by running the **SHOW ALL** statement.  
 **NOTE**  
Some parameters that viewed by **SHOW ALL** cannot be set by **SET**. For example, **max\_datanodes**.
- **CURRENT\_SCHEMA**  
Specifies the current schema.
- **TIME\_ZONE**  
Specifies the time zone.
- **TRANSACTION ISOLATION LEVEL**  
Specifies the transaction isolation level.
- **SESSION AUTHORIZATION**  
Specifies the current user identifier of the current session.
- **ALL**  
Resets all settable GUC parameters to default values.

### Examples

```
-- Reset timezone to the default value.
gaussdb=# RESET timezone;

-- Set all parameters to their default values.
gaussdb=# RESET ALL;
```

### Helpful Links

[SET](#) and [SHOW](#)

## 7.14.148 REVOKE

### Description

Revokes permissions from one or more roles.

### Precautions

If a non-owner user of an object attempts to the REVOKE permission on the object, the statement is executed based on the following rules:

- If the user has no permissions whatsoever on the object, the statement will fail outright.
- If an authorized user has some permissions, only the permissions with authorization options are revoked.
- If the authorized user does not have the authorization option, the **REVOKE ALL PRIVILEGES** form will issue an error message. For other forms of statements, if the permission specified in the statement does not have the corresponding authorization option, the statement will issue a warning.

### Syntax

- Revoke the permission on a specified table or view.

```
REVOKE [GRANT OPTION FOR]
 { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
 INDEX | VACUUM }, ... }
 | ALL [PRIVILEGES] }
ON { [TABLE] table_name [, ...]
 | ALL TABLES IN SCHEMA schema_name [, ...] }
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- Revoke the permission on a specified field in a table.

```
REVOKE [GRANT OPTION FOR]
 { { SELECT | INSERT | UPDATE | REFERENCES | COMMENT } (column_name [, ...]), ... }
 | ALL [PRIVILEGES] (column_name [, ...]) }
ON [TABLE] table_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- Revoke the permission on a specified sequence.

```
REVOKE [GRANT OPTION FOR]
 { { SELECT | UPDATE | ALTER | DROP | COMMENT }, ... }
 | ALL [PRIVILEGES] }
ON { [SEQUENCE] sequence_name [, ...]
 | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- Revoke the permission on a specified database.

```
REVOKE [GRANT OPTION FOR]
 { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
 | ALL [PRIVILEGES] }
ON DATABASE database_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- Revoke the permission on a specified domain.

```
REVOKE [GRANT OPTION FOR]
 { USAGE | ALL [PRIVILEGES] }
ON DOMAIN domain_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```

- **Revoke the specified CMK permission.**

```
REVOKE [GRANT OPTION FOR]
{ { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
ON CLIENT_MASTER_KEYS client_master_keys_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the specified CEK permission.**

```
REVOKE [GRANT OPTION FOR]
{ { USAGE | DROP } [, ...] | ALL [PRIVILEGES]}
ON COLUMN_ENCRYPTION_KEYS column_encryption_keys_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the permission on a specified directory.**

```
REVOKE [GRANT OPTION FOR]
{ { READ | WRITE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
ON DIRECTORY directory_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the permission on a specified external data source.**

```
REVOKE [GRANT OPTION FOR]
{ USAGE | ALL [PRIVILEGES] }
ON FOREIGN DATA WRAPPER fdw_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the permission on a specified external server.**

```
REVOKE [GRANT OPTION FOR]
{ { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON FOREIGN SERVER server_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the permission on a specified function.**

```
REVOKE [GRANT OPTION FOR]
{ { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON { FUNCTION {function_name ([{ [argmode] [arg_name] arg_type } [, ...]) } } [, ...]
| ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the permission on a specified stored procedure.**

```
REVOKE [GRANT OPTION FOR]
{ { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON { PROCEDURE {proc_name ([{ [argmode] [arg_name] arg_type } [, ...]) } } [, ...]
| ALL PROCEDURE IN SCHEMA schema_name [, ...] }
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the permission on a specified procedural language.**

```
REVOKE [GRANT OPTION FOR]
{ USAGE | ALL [PRIVILEGES] }
ON LANGUAGE lang_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the permission on a specified large object.**

```
REVOKE [GRANT OPTION FOR]
{ { SELECT | UPDATE } [, ...] | ALL [PRIVILEGES] }
ON LARGE OBJECT loid [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```
- **Revoke the permission on a specified schema.**

```
REVOKE [GRANT OPTION FOR]
{ { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
ON SCHEMA schema_name [, ...]
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT];
```

- Revoke the permission on a specified tablespace.  

```
REVOKE [GRANT OPTION FOR]
 { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
 ON TABLESPACE tablespace_name [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```
- Revoke the permission on a specified type.  

```
REVOKE [GRANT OPTION FOR]
 { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
 ON TYPE type_name [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```
- Revoke the permission on a specified sub-cluster.  

```
REVOKE [GRANT OPTION FOR]
 { { CREATE | USAGE | COMPUTE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
 ON NODE GROUP group_name [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

**NOTE**

When the **create** permission on a sub-cluster is revoked, the **usage** and **compute** permissions are revoked by default.

- Revoke the permission on a directory object.  

```
REVOKE [GRANT OPTION FOR]
 { { READ | WRITE } [, ...] | ALL [PRIVILEGES] }
 ON DIRECTORY directory_name [, ...]
 FROM {[GROUP] role_name | PUBLIC} [, ...]
 [CASCADE | RESTRICT];
```
- Revoke permissions from a role.  

```
REVOKE [ADMIN OPTION FOR]
 role_name [, ...] FROM role_name [, ...]
 [CASCADE | RESTRICT];
```
- Revoke the SYSADMIN permission from a role.  

```
REVOKE ALL { PRIVILEGES | PRIVILEGE } FROM role_name;
```
- Revoke the ANY permissions.  

```
REVOKE [ADMIN OPTION FOR]
 { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT ANY
 TABLE | UPDATE ANY TABLE |
 DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
 EXECUTE ANY FUNCTION |
 CREATE ANY TYPE | ALTER ANY TYPE | DROP ANY TYPE | ALTER ANY SEQUENCE | DROP ANY
 SEQUENCE |
 SELECT ANY SEQUENCE | ALTER ANY INDEX | DROP ANY INDEX | CREATE ANY SYNONYM | DROP
 ANY SYNONYM | CREATE ANY TRIGGER | ALTER ANY TRIGGER |
 DROP ANY TRIGGER
 } [, ...]
 FROM [GROUP] role_name [, ...];
```
- Revoke the permission on database links.  

```
REVOKE { CREATE | ALTER | DROP } [PUBLIC] DATABASE LINK FROM role_name;
```

**NOTE**

For details about database links, see [DATABASE LINK](#).

## Parameters

The keyword **PUBLIC** indicates an implicitly defined group that has all roles.

For details about permission types and parameters, see [Parameters](#) in "GRANT."

Permissions of a role include the permissions directly granted to the role, permissions inherited from the parent role, and permissions granted to **PUBLIC**.

Therefore, revoking the **SELECT** permission for an object from **PUBLIC** does not necessarily mean that the **SELECT** permission for the object has been revoked from all roles, because the **SELECT** permission directly granted to roles and inherited from parent roles remains. Similarly, if the **SELECT** permission is revoked from a user but is not revoked from **PUBLIC**, the user can still run the **SELECT** statement.

If **GRANT OPTION FOR** is specified, the permission cannot be granted to others, but permission itself is not revoked.

If user A holds the UPDATE permissions on a table, specifies the **WITH GRANT OPTION** option, and has granted the permissions to user B, the permissions that user B holds are called dependent permissions. If the permissions or the grant option held by user A is revoked, the dependent permissions still exist. Those dependent permissions are also revoked if **CASCADE** is specified.

A user can only revoke permissions that were granted directly by that user. For example, if user A has granted permission with grant option (**WITH ADMIN OPTION**) to user B, and user B has in turn granted it to user C, then user A cannot revoke the permission directly from C. However, user A can revoke the grant option held by user B and use **CASCADE**. In this way, the permission of user C is automatically revoked. For another example, if both user A and user B have granted the same permission to C, A can revoke his own grant but not B's grant, so C will still effectively have the permission.

If the role executing **REVOKE** holds permissions indirectly via more than one role membership path, it is unspecified which containing role will be used to execute the statement. In such cases, you are advised to use **SET ROLE** to become the specific role you want to do the **REVOKE** as, and then execute **REVOKE**. Failure to do so may lead to deleting permissions not intended to delete, or not deleting any permissions at all.

## Examples

See [Examples](#) in section "GRANT."

## Helpful Links

[GRANT](#)

## 7.14.149 ROLLBACK

### Description

Rolls back the current transaction and backs out all updates in the transaction.

If a fault occurs during the running of a transaction, the transaction cannot be executed. The system cancels all completed operations on the database in the transaction, and the database status returns to the time when the transaction starts.

### Precautions

If a **ROLLBACK** statement is executed out of a transaction, no error occurs, but a notice is displayed.

## Syntax

```
ROLLBACK [WORK | TRANSACTION];
```

## Parameters

- **WORK | TRANSACTION**  
Specifies the optional keyword that more clearly illustrates the syntax.

## Examples

```
-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Roll back all changes.
gaussdb=# ROLLBACK;
```

## Helpful Links

[COMMIT | END](#)

## 7.14.150 ROLLBACK PREPARED

### Description

Cancels a transaction ready for two-phase committing.

### Precautions

- The function is only available in maintenance mode (when the GUC parameter **xc\_maintenance\_mode** is set to **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the user who initiates a transaction or a system administrator can roll back the transaction.
- The transaction function is maintained automatically by the database, and should be not visible to users.

## Syntax

```
ROLLBACK PREPARED transaction_id ;
```

## Parameters

- **transaction\_id**  
Specifies the identifier of the transaction to be committed. The identifier must be different from those for current prepared transactions.

## Helpful Links

[COMMIT PREPARED](#) and [PREPARE TRANSACTION](#)

## 7.14.151 ROLLBACK TO SAVEPOINT

### Description

Rolls back to a savepoint. It implicitly destroys all savepoints that were established after the named savepoint.

Rolls back all statements that were executed after the savepoint was established. The savepoint remains valid and can be rolled back to again later, if needed.

### Precautions

- Specifying a savepoint name that has not been established is an error.
- Cursors have somewhat non-transactional behavior with respect to savepoints. Any cursor that is opened inside a savepoint will be closed when the savepoint is rolled back. If a previously opened cursor is affected by a FETCH statement inside a savepoint that is later rolled back, the cursor remains at the position that FETCH left it pointing to (that is, the cursor motion caused by FETCH is not rolled back). Closing a cursor is not undone by rolling back, either. A cursor whose execution causes a transaction to abort is put in a cannot-execute state, so while the transaction can be restored using ROLLBACK TO SAVEPOINT, the cursor can no longer be used.
- Use ROLLBACK TO SAVEPOINT to roll back to a savepoint. Use RELEASE SAVEPOINT to destroy a savepoint but keep the effects of the statements executed after the savepoint was established.

### Syntax

```
ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT] savepoint_name;
```

### Parameters

*savepoint\_name*

Rolls back to a savepoint.

### Examples

```
-- Undo the effects of the statements executed after my_savepoint was established:
gaussdb=# START TRANSACTION;
gaussdb=# SAVEPOINT my_savepoint;
gaussdb=# ROLLBACK TO SAVEPOINT my_savepoint;
-- Cursor positions are not affected by savepoint rollback.
gaussdb=# DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
gaussdb=# SAVEPOINT foo;
gaussdb=# FETCH 1 FROM foo;
?column?

1
gaussdb=# ROLLBACK TO SAVEPOINT foo;
gaussdb=# FETCH 1 FROM foo;
?column?

2
gaussdb=# RELEASE SAVEPOINT my_savepoint;
gaussdb=# COMMIT;
```

## Helpful Links

[SAVEPOINT](#) and [RELEASE SAVEPOINT](#)

### 7.14.152 SAVEPOINT

#### Description

SAVEPOINT establishes a new savepoint in the current transaction.

A savepoint is a special mark inside a transaction. It allows all statements that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint.

#### Precautions

- Use ROLLBACK TO SAVEPOINT to roll back to a savepoint. Use RELEASE SAVEPOINT to destroy a savepoint but keep the effects of the statements executed after the savepoint was established.
- Savepoints can only be established when inside a transaction block. Multiple savepoints can be defined in a transaction.
- In the case of an unexpected termination of a distributed thread or process caused by a node or connection failure, or of an error caused by the inconsistency between source and destination table structures in a COPY FROM operation, the transaction cannot be rolled back to the established savepoint. Instead, the entire transaction will be rolled back.
- According to the SQL standard, when a savepoint with the same name is created, the previous savepoint with the same name is automatically deleted. In GaussDB, the old savepoint is retained, but only the latest one is used during rollback or release. If the latest savepoint is released, the previous savepoint will again become accessible to ROLLBACK TO SAVEPOINT and RELEASE SAVEPOINT. In addition, SAVEPOINT fully complies with the SQL standard.

#### Syntax

```
SAVEPOINT savepoint_name;
```

#### Parameters

savepoint\_name

Specifies the name of the new savepoint.

---

#### NOTICE

When using SAVEPOINT, you are advised to release SAVEPOINT promptly to avoid too many nested subtransactions. It is recommended that the number of nested subtransactions be less than or equal to 10,000. If the number of nested subtransactions is too large, the current transaction performance may deteriorate.

---

## Examples

```
-- Create a table.
gaussdb=# CREATE TABLE table1(a int);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO table1 VALUES (1);

-- Create a savepoint.
gaussdb=# SAVEPOINT my_savepoint;

-- Insert data.
gaussdb=# INSERT INTO table1 VALUES (2);

-- Roll back the savepoint.
gaussdb=# ROLLBACK TO SAVEPOINT my_savepoint;

-- Insert data.
gaussdb=# INSERT INTO table1 VALUES (3);

-- Commit the transaction.
gaussdb=# COMMIT;

-- Query the content of the table. You can see 1 and 3 at the same time, but cannot see 2 because 2 is
rolled back.
gaussdb=# SELECT * FROM table1;

-- Delete the table.
gaussdb=# DROP TABLE table1;

-- Create a table.
gaussdb=# CREATE TABLE table2(a int);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO table2 VALUES (3);

-- Create a savepoint.
gaussdb=# SAVEPOINT my_savepoint;

-- Insert data.
gaussdb=# INSERT INTO table2 VALUES (4);

-- Roll back the savepoint.
gaussdb=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
gaussdb=# COMMIT;

-- Query the table content. You can see 3 and 4 at the same time.
gaussdb=# SELECT * FROM table2;

-- Delete the table.
gaussdb=# DROP TABLE table2;
```

## Helpful Links

[RELEASE SAVEPOINT](#) and [ROLLBACK TO SAVEPOINT](#)

## 7.14.153 SELECT

### Description

SELECT retrieves data from tables or views.

Serving as an overlaid filter on database tables, the SELECT statement uses SQL keywords to filter data tables and extract the required data.

### Precautions

- The owner of a table, users with the SELECT permission on the table, or users with the SELECT ANY TABLE permission can read data in the table or view. System administrators have the permission to read data in the table or view by default.
- SELECT can join ordinary tables, but cannot join ordinary and GDS foreign tables. That is, the SELECT statement cannot contain both an ordinary table and a GDS foreign table.
- You must have the SELECT permission on each field used in the SELECT statement.
- To use FOR UPDATE or FOR SHARE, you must have the UPDATE permission in addition to the SELECT permission.

### Syntax

- Query data.

```
[WITH [RECURSIVE] with_query [, ...]]
SELECT [/*+ plan_hint */] [ALL | DISTINCT [ON (expression [, ...])]]
 { * | {expression [[AS] output_name]} [, ...] }
 [FROM from_item [, ...]]
 [WHERE condition]
 [GROUP BY grouping_element [, ...]]
 [HAVING condition [, ...]]
 [WINDOW {window_name AS (window_definition)} [, ...]]
 [{ UNION | INTERSECT | EXCEPT | MINUS } [ALL | DISTINCT] select]
 [ORDER BY {expression [[ASC | DESC | USING operator] | nlssort_expression_clause] [NULLS { FIRST |
LAST }]} [, ...]]
 [LIMIT { [offset,] count | ALL }]
 [OFFSET start [ROW | ROWS]]
 [FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY]
 [{FOR { UPDATE | SHARE } [OF table_name [, ...]] [NOWAIT | WAIT n]} [...]]
TABLE { ONLY { (table_name) | table_name } | table_name [*]};
```

#### NOTE

In condition and expression, you can use the aliases of expressions in **targetlist** in compliance with the following rules:

- Reference only within the same level.
- Reference only the aliases in the **targetlist**.
- Reference a prior expression in a subsequent expression.
- Volatile functions cannot be used.
- Window functions cannot be used.
- Aliases cannot be referenced in the condition of JOIN ON.
- An error is reported if **targetlist** contains multiple aliases to be referenced.

## NOTICE

In the scenario where the SELECT statement plan is cached, the WHERE IN candidate subset cannot be too large. It is recommended that the number of conditions be less than or equal to 100 to prevent high dynamic memory.

- If the WHERE IN candidate subset is too large, the memory usage of the generated plan increases.
- If the WHERE IN subsets constructed by concatenated SQL statements are different, the SQL template of the cache plan cannot be reused. A large number of different plans are generated, and the plans cannot be shared. As a result, a large amount of memory is occupied.

- The subquery **with\_query** is as follows:

```
with_query_name [(column_name [, ...])]
AS [[NOT] MATERIALIZED] ({select | values | insert | update | delete})
```

- The specified query source **from\_item** is as follows:

```
{[ONLY] table_name [*] [partition_clause] [[AS] alias [(column_alias [, ...])]]
[TABLESAMPLE sampling_method (argument [, ...]) [REPEATABLE (seed)]]
{ (select) [AS] alias [(column_alias [, ...])]
|with_query_name [[AS] alias [(column_alias [, ...])]]
|function_name ([argument [, ...]]) [AS] alias [(column_alias [, ...] | column_definition [, ...])]
|function_name ([argument [, ...]]) AS (column_definition [, ...])
|from_item unpivot_clause
|from_item pivot_clause
|from_item [NATURAL] join_type from_item [ON join_condition | USING (join_column [, ...])] }
```

- The GROUP clause is as follows:

```
()
| expression
| (expression [, ...])
| ROLLUP ({ expression | (expression [, ...]) } [, ...])
| CUBE ({ expression | (expression [, ...]) } [, ...])
| GROUPING SETS (grouping_element [, ...])
```

- The specified partition **partition\_clause** is as follows:

```
PARTITION { (partition_name) |
FOR (partition_value [, ...]) }
```

### NOTE

Specifying partitions applies only to partitioned tables.

- The sorting order **nlsort\_expression\_clause** is as follows:  
NLSORT ( column\_name, ' NLS\_SORT = { SCHINESE\_PINYIN\_M | generic\_m\_ci } ' )
- Simplified query syntax, equivalent to SELECT \* FROM *table\_name*.  
TABLE { ONLY {(table\_name)| table\_name} | table\_name [ \* ]};

## Parameters

- **WITH [ RECURSIVE ] with\_query [, ...]**

Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table. This subquery statement structure is called the common table expression (CTE) structure. When this structure is used, the execution plan contains the CTE SCAN content.

If **RECURSIVE** is specified, it allows a **SELECT** subquery to reference itself by name.

The detailed format of **with\_query** is as follows: **with\_query\_name**  
[ ( column\_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )

- **with\_query\_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.
- **column\_name** specifies the column name displayed in the subquery result set.
- Each subquery can be a SELECT, VALUES, INSERT, UPDATE, or DELETE statement.
- RECURSIVE can appear only after WITH. In the case of multiple CTEs, you only need to declare RECURSIVE at the first CTE.
- You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE. Currently, only inline execution is supported for stream plans. In this case, this syntax does not take effect.
  - If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.
  - If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the SELECT statement trunk to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.
- **plan\_hint** clause  
Follows the SELECT keyword in the */\*+<Plan hint> \*/* format. It is used to optimize the plan of a SELECT statement block. For details, see [Hint-based Tuning](#). In each statement, only the first */\*+ plan\_hint \*/* comment block takes effect as a hint. Multiple hints can be written.
- **ALL**  
Specifies that all rows that meet the conditions are returned. This is the default behavior and can be omitted.
- **DISTINCT [ ON ( expression [, ...] ) ]**  
Removes all duplicate rows from the **SELECT** result set so one row is kept from each group of duplicates.  
Retains only the first row in the set of rows that have the same result calculated on the given expression.

---

**NOTICE**

**DISTINCT ON** expression is explained with the same rule of **ORDER BY**. Unless you use ORDER BY to guarantee that the required row appears first, you cannot know what the first row is.

---

- **SELECT list**

Specifies the name of a column in the table to be queried. The value can be a part of the column name or all of the column names. The wildcard (\*) is used to represent the column name.

You may use the AS *output\_name* clause to assign an alias to the output column. The alias is usually used for displaying the output column. The name, value, and type keywords can be used as column aliases.

Column names can be expressed in the following formats:

- Manually input column names which are spaced using commas (,).
- Columns computed in the FROM clause.

- **FROM clause**

Specifies one or more source tables for SELECT.

The FROM clause can contain the following elements:

- *table\_name*

Specifies the name of a table or view. The schema name can be added before the table name or view name, for example, *schema\_name.table\_name*.

 **NOTE**

You can use database link to perform operations on remote tables and synonyms. For details, see [DATABASE LINK](#).

- *alias*

Gives a temporary alias to a table to facilitate the reference by other queries.

An alias is used for brevity or to eliminate ambiguity for self-joins. If an alias is provided, it completely replaces the actual name of the table.

---

**NOTICE**

If an alias is specified for the *joined\_table* table created by JOIN and *joined\_table* is wrapped with "()", that is, (*joined\_table*), non-reserved keywords UNPIVOT and PIVOT cannot be used as aliases.

- TABLESAMPLE *sampling\_method* ( *argument* [, ...] ) [ REPEATABLE ( *seed* ) ]

The TABLESAMPLE clause following **table\_name** specifies that the specified **sampling\_method** should be used to retrieve the subset of rows in the table.

The optional REPEATABLE clause specifies the number of *seeds* used to generate random numbers in the sampling method. The seed value can be any non-null constant value. If the table was not changed during the query, the two queries having the same seed and **argument** values will select the same sampling in this table. However, different seed values usually generate different samples. If **REPEATABLE** is not specified, a new random sample will be selected for each query based on the seed generated by the system.

- TIMECAPSULE { **TIMESTAMP** | **CSN** } *expression*

Queries the table data of a specified CSN or at a specified time point.

Currently, the following tables do not support flashback query: system catalogs, DFS tables, global temporary tables, local temporary tables, unlogged tables, views, sequence tables, hash bucket tables, shared tables, and inherited tables.

- TIMECAPSULE TIMESTAMP

Searches for the result set of a specified time point based on the date as the flashback query flag. *date* must be a valid past timestamp

- expression

Specifies constants, functions, or SQL expressions.

- TIMECAPSULE CSN

Searches for the result set of a specified CSN based on the CSN flashback of the table as the flashback query flag. The CSN can be obtained from **snpcsn** recorded in **gs\_txn\_snapshot**.

 NOTE

- A flashback query cannot span statements that affect the table structure or physical storage. Otherwise, an error is reported. Between the flashback point and the current point, if a statement (TRUNCATE, DDL, DCL, or VACUUM FULL) has been executed to modify the table structure or affect physical storage, the flashback fails. The error message "ERROR: The table definition of %s has been changed." is displayed when flashback is performed on a table where DDL operations have been performed. The error message "ERROR: recycle object %s desired does not exist;" is displayed when flashback is performed on a table where DDL operations, such as changing namespaces and table names, have been performed.
  - Flashback query does not support index query. Flashback query supports only seqScan for full table scanning.
  - When the flashback point is too old, the source version cannot be obtained because the flashback version is recycled. As a result, the flashback fails and the error message "Restore point too old" is displayed.
  - The flashback point is specified by time. The maximum difference between the flashback point and the actual time is 3 seconds.
  - After truncating a table, perform a flashback query or flashback on the table. The error message "Snapshot too old" is displayed when a flashback is performed at a specified time point. Data cannot be found or the error message "Snapshot too old" is reported during the CSN-based flashback.
  - In the GTM-Free scenario, each node uses the local CSN and does not have a globally unified CSN. Therefore, flashback in CSN mode is not supported.
- column\_alias  
Specifies the column alias.
  - PARTITION  
Queries data in the specified partition in a partitioned table.
  - partition\_name  
Specifies the name of a partition.

- partition\_value  
Specifies the value of the specified partition key. If a partitioned table has multiple partition keys, you can use PARTITION FOR to uniquely identify a partition.
- subquery  
Performs a subquery in the FROM clause. A temporary table is created to save the output of the subquery.
- with\_query\_name  
References a WITH clause as the source of a FROM clause by the name of a WITH query.
- function\_name  
Function name. Functions can be used in a FROM clause.
- join\_type  
The options are as follows:
  - [ INNER ] JOIN  
A JOIN clause combines two FROM items. You can use parentheses to determine the order of nesting. In the absence of parentheses, **JOIN** nests left-to-right.
  - LEFT [ OUTER ] JOIN  
Returns all rows that meet the join conditions in the Cartesian product, as well as the rows in the left table that do not match the right table rows according to join conditions. This left-hand row is extended to the full width of the joined table by inserting **NULL** values for the right-hand columns. Note that the condition of the JOIN clause is used only when matches are being calculated. The condition of the outer layer is applied after the calculation.
  - RIGHT [ OUTER ] JOIN  
Returns all result rows of INNER JOIN, as well as all rows on the right that do not have matches (rows on the left are filled with **NULL**).  
This is just a notational convenience, since you could convert it to a **LEFT OUTER JOIN** by switching the left and right inputs.
  - FULL [ OUTER ] JOIN  
Returns all the joined rows, pluses one row for each unmatched left-hand row (extended with **NULL** on the right), and pluses one row for each unmatched right-hand row (extended with **NULL** on the left).
  - CROSS JOIN  
Is equivalent to **INNER JOIN ON (TRUE)**, which means no rows are removed by qualification. These join types are just a notational convenience, since they do nothing you could not do with plain **FROM** and **WHERE**.

 NOTE

For the **INNER** and **OUTER** join types, a join condition must be specified, namely exactly one of **NATURAL ON**, **join\_condition**, or **USING (join\_column [, ...])**. For **CROSS JOIN**, none of these clauses can appear.

**CROSS JOIN** and **INNER JOIN** produce a simple Cartesian product, the same result as you get from listing the two items at the top level of **FROM**.

- ON join\_condition

Defines which rows have matches in joins. Example: **ON left\_table.a = right\_table.a**. You are advised not to use numeric types such as int for **join\_condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

- USING(join\_column[, ...])

ON left\_table.a = right\_table.a AND left\_table.b = right\_table.b ... abbreviation. Corresponding columns must have the same name.

- NATURAL

Shorthand for a USING list that contains all columns in the two tables that have the same names.

- from item

Specifies the name of the query source object connected.

- unpivot\_clause

Converts a column to a row. The syntax format is as follows:

```
UNPIVOT [{INCLUDE | EXCLUDE} NULLS]
(
 unpivot_col_clause
 unpivot_for_clause
 unpivot_in_clause
)
```

▪ {INCLUDE | EXCLUDE} NULLS

Controls whether the converted result contains NULL rows. **INCLUDE NULLS** indicates that the converted results contain NULL rows. **EXCLUDE NULLS** indicates that the converted results do not contain NULL rows. If this clause is ignored, the unpivot operation removes NULL rows from the converted results by default.

▪ unpivot\_col\_clause

unpivot\_col\_element

**unpivot\_col\_element** specifies the output column names. These columns store the values to be converted.

▪ unpivot\_col\_element

{ column | ( column [, column]... ) }

unpivot\_col\_element has two forms: column and ( column [, column]... ).

▪ unpivot\_for\_clause

FOR { unpivot\_col\_element }

**unpivot\_col\_element** contained in the **unpivot\_for\_clause** is used to specify the output column names. These columns store the alias and names of the column to be converted.

- **unpivot\_in\_clause**  
IN ( unpivot\_in\_element [,unpivot\_in\_element...] )  
**unpivot\_in\_clause** specifies the columns to be converted. The column names and column values are saved in the previously specified output columns.  
**unpivot\_in\_element**  
{ unpivot\_col\_element }[ AS { unpivot\_alias\_element } ]  
**unpivot\_col\_element** specifies the column to be converted. If ( column [, column]...) is used to specify the column to be converted, the *column* names are connected by underscores ( \_ ) and saved in the output columns. For example, IN ((col1, col2)) generates the column name **col1\_col2** and saves it in the output column specified by **unpivot\_for\_clause**. In addition, the AS keyword can be used to specify an alias for the column to be converted. Once an alias is specified, the alias is saved in the output column instead of the name of column to be converted.
- **unpivot\_alias\_element**  
{ alias | ( alias [, alias]... ) }  
Similar to **unpivot\_col\_element**, **unpivot\_alias\_element** has two forms. **alias** indicates the specified alias.

---

**NOTICE**

Currently, **unpivot\_clause** has the following restrictions:

- This parameter can be used only in ORA compatibility mode.
- The **unpivot\_clause** cannot be used with hints.
- For **unpivot\_col\_clause**, the number of output columns specified by **unpivot\_col\_element** must be the same as that of **unpivot\_col\_element** contained in **unpivot\_in\_clause**.
- For **unpivot\_for\_clause**, the number of output columns specified by **unpivot\_col\_element** must be the same as the number of aliases specified by **unpivot\_alias\_element** contained in **unpivot\_in\_clause**.
- For **unpivot\_in\_clause**, the alias must be a constant or an expression that can be converted to a constant.
- For **unpivot\_in\_clause**, constant expressions support only IMMUTABLE functions.
- For all **unpivot\_col\_element** parameters contained in **unpivot\_in\_clause**, if the column types in the same position of these **unpivot\_col\_element** parameters are different, UNPIVOT attempts to convert the column types in order to convert the values of columns to be converted to a public type. Similarly, for all **unpivot\_alias\_element** parameters, if the alias types of these **unpivot\_alias\_element** parameters in the same position are different, UNPIVOT attempts to do the similar type conversion.

For example, assume that there is an **unpivot\_in\_clause** in the form of IN (col1, col2), where col1 is of the int type and col2 is of the float type, UNPIVOT attempts to convert the column value of col1 to the public type float during the calculation.

- pivot\_clause

Converts a row into column. The syntax format is as follows:

```
PIVOT [XML]
(aggregate_function (expr) [[AS] alias]
 [, aggregate_function (expr) [[AS] alias]]...
 pivot_for_clause
 pivot_in_clause
)
```

▪ aggregate\_function ( expr ) [[AS] alias ]

Aggregates calculation on a given expression. The calculation result is saved in the output column specified by **pivot\_in\_clause**. [AS] alias (The AS keyword can be omitted.) can be used to specify an alias for **aggregate\_function**. The alias is appended to the output column name specified by **pivot\_in\_clause** in the format of "\_alias".

▪ pivot\_for\_clause

```
FOR { column
 | (column [, column]...)
}
```

Specifies the row to be converted. The **column** indicates a column of the row to be converted.

▪ pivot\_in\_clause

```
IN ({ { { expr
 | (expr [, expr]...)
 } [[AS] alias]
 } ...
 }
)
```

Specifies the name of the output column. The column name can consist of one or more expressions, for example, (expr1, expr2). When a column name consists of multiple expressions, these expressions are connected by underscores (\_) in sequence. That is, the output column name corresponding to (expr1, expr2) is "expr1\_expr2". These expressions not only generate output column names, but also determine the time when the aggregate function is triggered. If the values of row to be converted is the same as the value of these expressions, the results calculated by **aggregate\_function** are saved in the output column names that consist of these expressions. Assume that expr1 is "1" and expr2 is "2". For row "1 2", **aggregate\_function** is used for calculation. For row "1 1", the calculation is not triggered.

#### NOTICE

Currently, **pivot\_clause** has the following restrictions:

- This parameter can be used only in ORA compatibility mode.
- **pivot\_clause** cannot be used with hints.
- If more than one **aggregate\_function** is specified, at most one **aggregate\_function** is allowed to have no alias, the rest of the **aggregate\_function** functions are required to specify an alias.
- XML supports only syntax but does not support functions.
- The expression in **pivot\_in\_clause** can be a constant or an expression that can be converted to a constant. If the expression is not a unary expression, specify an alias for the expression.
- For **pivot\_in\_clause**, constant expressions support only IMMUTABLE functions.
- For the expression in **pivot\_in\_clause**, when the keyword AS is used to specify an alias for the expression, only the non-reserved keywords can be used as aliases.
- If the length of an output column name exceeds 63 characters, subsequent characters will not be printed.

- **WHERE clause**

The WHERE clause forms an expression for row selection to narrow down the query range of SELECT. **condition** indicates any expression that returns a value of Boolean type. Rows that do not meet this condition will not be retrieved. You are advised not to use numeric types such as int for **condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

In a WHERE clause, you can use the operator (+) to convert a table join to an outer join. However, this method is not recommended because it is not the standard SQL syntax and may raise syntax compatibility issues during platform migration. There are many restrictions on using the operator (+):

- a. It can appear only in the WHERE clause.
- b. If a table join has been specified in the FROM clause, the operator (+) cannot be used in the WHERE clause.
- c. The operator (+) can work only on columns of tables or views, instead of on expressions.
- d. If table A and table B have multiple join conditions, the operator (+) must be specified in all the conditions. Otherwise, the operator (+) will not take effect, and the table join will be converted into an inner join without any prompt information.
- e. Tables specified in a join condition where the operator (+) works cannot cross queries or subqueries. If tables where the operator (+) works are not in the FROM clause of the current query or subquery, an error will be reported. If a peer table for the operator (+) does not exist, no error will be reported and the table join will be converted into an inner join.
- f. Expressions where the operator (+) is used cannot be directly connected through **OR**.

- g. If a column where the operator (+) works is compared with a constant, the expression becomes a part of the join condition.
- h. A table cannot have multiple foreign tables.
- i. The operator (+) can appear only in the following expressions: comparison, NOT, ANY, ALL, IN, NULLIF, IS DISTINCT FROM, and IS OF. It is not allowed in other types of expressions. In addition, these expressions cannot be connected through **AND** or **OR**.
- j. The operator (+) can be used to convert a table join only to a left or right outer join, instead of a full join. That is, the operator (+) cannot be specified on both tables of an expression.

---

#### NOTICE

For the WHERE clause, if special character "%", "\_", or "\" is queried in LIKE, add the slash "\" before each character.

---

- **GROUP BY clause**

Condenses query results into a single row all selected rows that share the same values for the grouped expressions.

- CUBE ( { expression | ( expression [, ...] ) } [, ...] )

A CUBE grouping is an extension to the GROUP BY clause that creates subtotals for all of the possible combinations of the given list of grouping columns (or expressions). In terms of multidimensional analysis, CUBE generates all the subtotals that could be calculated for a data cube with the specified dimensions. For example, given three expressions ( $n=3$ ) in the CUBE clause, the operation results in  $2^n = 2^3 = 8$  groupings. Rows grouped on the values of  $n$  expressions are called regular rows, and the rest are called super-aggregate rows.

- GROUPING SETS ( grouping\_element [, ...] )

GROUPING SETS is another extension to the GROUP BY clause. It allows you to specify multiple GROUP BY clauses. This improves efficiency by trimming away unnecessary data. After you specify the required data group, the database does not need to compute a whole **ROLLUP** or **CUBE**.

---

#### NOTICE

- If a SELECT list expression references ungrouped columns and no aggregate function is used, an error is reported. This is because multiple values may be returned for ungrouped columns.
  - If a SELECT list expression references a constant, the GROUP BY clause does not need to group the constant. Otherwise, an error is reported.
  - When a primary key is used in the GROUP BY clause, all non-aggregated columns in the SELECT statement can appear in the result set because the uniqueness of the primary key ensures the certainty of non-aggregated column values in each group.
-

- **HAVING clause**

Used together with the GROUP BY clause to select special groups. The HAVING clause compares some attributes of groups with a constant. Only groups that match the logical expression in the HAVING clause are extracted.

- **WINDOW clause**

The general format is **WINDOW window\_name AS ( window\_definition ) [ , ... ]**. **window\_name** is a name can be referenced by **window\_definition**. **window\_definition** can be expressed in the following forms:

```
[existing_window_name]
[PARTITION BY expression [, ...]]
[ORDER BY expression [ASC | DESC | USING operator] [NULLS { FIRST | LAST }] [, ...]]
[frame_clause]
```

**frame\_clause** defines a **window frame** for the window function. The window function (not all window functions) depends on **window frame** and **window frame** is a set of relevant rows of the current query row. **frame\_clause** can be expressed in the following forms:

```
[RANGE | ROWS] frame_start
[RANGE | ROWS] BETWEEN frame_start AND frame_end
frame_start and frame_end can be expressed in the following forms:
UNBOUNDED PRECEDING
value PRECEDING
CURRENT ROW
value FOLLOWING
UNBOUNDED FOLLOWING
```

 **NOTE**

Currently, **frame\_start** and **frame\_end** do not support **value PRECEDING** and **value FOLLOWING**.

- **UNION clause**

Computes the set union of the rows returned by multiple SELECT statements. The UNION clause has the following constraints:

- By default, the result of UNION does not contain any duplicate rows unless the ALL clause is declared.
- UNION operators in the same SELECT statement are evaluated from left to right unless specified by parentheses.
- **FOR UPDATE** cannot be specified either for a UNION result or for any input of a UNION.

General expression:

```
select_statement UNION [ALL] select_statement
```

- **select\_statement** can be any SELECT statement without an ORDER BY, LIMIT, or FOR UPDATE clause.
- ORDER BY and LIMIT can be attached to subexpressions if enclosed in parentheses.

---

**NOTICE**

It is recommended that the number of UNION ALL clauses be less than 100. If the number of UNION ALL clauses exceeds 100, ensure that the instance memory is sufficient.

---

- **INTERSECT clause**

Computes the set intersection of rows returned by the involved SELECT statements. The result of INTERSECT does not contain any duplicate rows.

The INTERSECT clause has the following constraints:

- Multiple INTERSECT operators in the same SELECT statement are evaluated left to right, unless otherwise specified by parentheses.
- Processing INTERSECT preferentially when UNION and INTERSECT operations are executed for results of multiple SELECT statements.

General format:

```
select_statement INTERSECT select_statement
```

**select\_statement** can be any SELECT statement without a FOR UPDATE clause.

- **EXCEPT clause**

Has the following common form:

```
select_statement EXCEPT [ALL] select_statement
```

**select\_statement** can be any SELECT statement without a FOR UPDATE clause.

The EXCEPT operator computes the set of rows that are in the result of the left SELECT statement but not in the result of the right one.

The result of EXCEPT does not contain any duplicate rows unless the ALL clause is declared. To execute **ALL**, a row that has  $m$  duplicates in the left table and  $n$  duplicates in the right table will appear  $\text{MAX}(m-n, 0)$  times in the result set.

Multiple **EXCEPT** operators in the same **SELECT** statement are evaluated left to right, unless parentheses dictate otherwise. **EXCEPT** binds at the same level as **UNION**.

Currently, **FOR UPDATE** cannot be specified either for an **EXCEPT** result or for any input of an **EXCEPT**.

- **MINUS clause**

Has the same function and syntax as EXCEPT clause.

- **ORDER BY clause**

Sorts data retrieved by SELECT in descending or ascending order. If the ORDER BY expression contains multiple columns:

- If two columns are equal according to the leftmost expression, they are compared according to the next expression and so on.
- If they are equal according to all specified expressions, they are returned in an implementation-dependent order.
- When used with the DISTINCT keyword, the columns to be sorted in ORDER BY must be included in the columns of the result set retrieved by the SELECT statement.
- When used with the GROUP BY clause, the columns to be sorted in ORDER BY must be included in the columns of the result set retrieved by the SELECT statement.

## NOTICE

To support Chinese pinyin order, set the encoding format to **UTF-8**, **GB18030**, **GB18030\_2022**, or **GBK** when initializing the database. The statements are as follows:

```
initdb -E UTF8 -D ../data -locale=zh_CN.UTF-8, initdb -E GB18030 -D ../data -locale=zh_CN.GB18030, initdb -E GB18030_2022 -D ../data -locale=zh_CN.GB18030, or initdb -E GBK -D ../data -locale=zh_CN.GBK.
```

- **LIMIT clause**

Consists of two independent sub-clauses:

**LIMIT { count | ALL }** limits the number of rows to be returned. **count** specifies the number of rows to be returned. The effect of **LIMIT ALL** is the same as that of omitting the LIMIT clause.

**OFFSET start count** specifies the maximum number of rows to return, while **start** specifies the number of rows to skip before starting to return rows. When both are specified, **start** rows are skipped before starting to count the **count** rows to be returned.

ROWNUM cannot be used as count or offset in the LIMIT clause.

- **OFFSET clause**

The SQL: 2008 standard has introduced a different clause:

**OFFSET start { ROW | ROWS }**

**start** specifies the number of rows to skip before starting to return rows.

- **FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY**

If **count** is omitted in a FETCH clause, it defaults to **1**.

- **FOR UPDATE clause**

The FOR UPDATE clause locks the rows retrieved by SELECT. This ensures that the rows cannot be modified or deleted by other transactions until the current transaction ends. That is, other transactions that attempt UPDATE, DELETE, or SELECT FOR UPDATE of these rows will be blocked until the current transaction ends.

To prevent the operation from waiting for the commit of other transactions, you can use **NOWAIT**. If the selected row cannot be locked immediately, an error is reported immediately when you execute **SELECT FOR UPDATE NOWAIT**. If you use **WAIT n** and the selected row cannot be locked immediately, the operation needs to wait for *n* seconds (the value of *n* is of the int type with a range of  $0 \leq n \leq 2147483$ ). If the lock is obtained within *n* seconds, the operation is performed normally. Otherwise, an error is reported.

**FOR SHARE** behaves similarly, except that it acquires a shared rather than exclusive lock on each retrieved row. A shared lock blocks other transaction from executing **UPDATE**, **DELETE**, or **SELECT FOR UPDATE** on these rows, but it does not prevent them from executing **SELECT FOR SHARE**.

If specified tables are named as specified by **FOR UPDATE** or **FOR SHARE**, then only rows coming from those tables are locked. Any other tables used in SELECT are simply read as usual. Otherwise, locking all tables in the statement.

If FOR UPDATE or FOR SHARE is applied to a view or sub-query, it affects all tables used in the view or sub-query.

Multiple FOR UPDATE and FOR SHARE clauses can be written if it is necessary to specify different locking behaviors for different tables.

If the same table is mentioned (or implicitly affected) by both FOR UPDATE and FOR SHARE clauses, it is processed as FOR UPDATE. Similarly, a table is processed as **NOWAIT** if that is specified in any of the clauses affecting it.

---

#### NOTICE

- For SQL statements containing FOR UPDATE or FOR SHARE, their execution plans will be pushed down to DNs. If the pushdown fails, an error will be reported.
- The query of row number contained in the projection column or WHERE conditions does not support FOR UPDATE/SHARE.
- For the FOR UPDATE/SHARE statements whose subquery is a stream plan, the same locked row cannot be concurrently updated.
- For the ORDER BY FOR UPDATE/SHARE statement, the execution sequence of the SORT and LOCKROW operators in the stream plan is different from that in other plans. In the stream plan, LOCK is executed before SORT. In other plans, SORT is executed before LOCK. The reason is that if the data in the stream plan is not on the current DN, the data needs to be redistributed and locked on the original DN. After redistribution, data becomes disordered. Therefore, the SORT operator needs to be added. If SORT is performed before LOCK, the original ordered data becomes disordered again. In this case, the SORT operator is meaningless and can be eliminated. The final plan execution sequence is changed from **sort > lock > sort** to **lock > sort > lock > sort**.
- In scenarios where FOR UPDATE/SHARE statements are concurrently executed, ORDER BY is used to sort data to avoid deadlocks. This method is not feasible for distributed systems because the DN lock sequence cannot be ensured by ORDER BY. In addition, adding ORDER BY causes performance overhead. Therefore, you are advised not to add ORDER BY to solve the deadlock problem.

---

- **NLS\_SORT**

Specifies that a column is sorted in a special order. Currently, only Chinese Pinyin and case-insensitive sorting are supported. To support this sorting mode, you need to set the encoding format to UTF8, GB18030, GB18030\_2022, or GBK when creating a database. If you set the encoding format to another format, for example, SQL\_ASCII, an error may be reported or the sorting mode may be invalid.

Value range:

- **SCHINESE\_PINYIN\_M**, sorted by Pinyin order.
- **generic\_m\_ci**: sorted in case-insensitive order (optional; only English characters are supported in the case-insensitive order.)

- **PARTITION clause**

Queries data in the specified partition in a partitioned table.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.reason values(3,'AAAAAAAABAAAAAAA','reason 1'),
(10,'AAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAABAAAAAAA','reason 3'),
(10,'AAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAABAAAAAAA','reason 5'),
(20,'AAAAAAACAAAAAAA','N%reason 6'),(30,'AAAAAAACAAAAAAA','W%reason 7');

-- Obtain the temp_t temporary table by a subquery and query all records in this table.
gaussdb=# WITH temp_t(name,isdba) AS (SELECT username,usesuper FROM pg_user) SELECT * FROM
temp_t;

-- Query all r_reason_sk records in the tpcds.reason table and delete duplicate records:
gaussdb=# SELECT DISTINCT(r_reason_sk) FROM tpcds.reason;

-- Example of a LIMIT clause: Obtain a record from the table.
gaussdb=# SELECT * FROM tpcds.reason LIMIT 1;

-- Query all records and sort them in alphabetic order.
gaussdb=# SELECT r_reason_desc FROM tpcds.reason ORDER BY r_reason_desc;

-- Use table aliases to obtain data from the pg_user and pg_user_status tables:
gaussdb=# SELECT a.username,b.locktime FROM pg_user a,pg_user_status b WHERE a.usesysid=b.roloid;

-- Example of the FULL JOIN clause: Join data in pg_user and pg_user_status.
gaussdb=# SELECT a.username,b.locktime,a.usesuper FROM pg_user a FULL JOIN pg_user_status b on
a.usesysid=b.roloid;

-- Example of the GROUP BY clause: Filter data based on query conditions, and group the results.
gaussdb=# SELECT r_reason_id, AVG(r_reason_sk) FROM tpcds.reason GROUP BY r_reason_id HAVING
AVG(r_reason_sk) > 25;

-- Example of the GROUP BY CUBE clause: Filter data based on query conditions, and group the results.
gaussdb=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY
CUBE(r_reason_id,r_reason_sk);

-- Example of the GROUP BY GROUPING SETS clause: Filter data based on query conditions, and group the
results.
gaussdb=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY GROUPING
SETS((r_reason_id,r_reason_sk),r_reason_sk);

-- Example of the UNION clause: Merge the names started with W and N in the r_reason_desc column in
the tpcds.reason table.
gaussdb=# SELECT r_reason_sk, tpcds.reason.r_reason_desc
FROM tpcds.reason
WHERE tpcds.reason.r_reason_desc LIKE 'W%'
UNION
SELECT r_reason_sk, tpcds.reason.r_reason_desc
FROM tpcds.reason
WHERE tpcds.reason.r_reason_desc LIKE 'N%';

-- Example of the NLS_SORT clause: Sort by Chinese Pinyin.
gaussdb=# SELECT * FROM tpcds.reason ORDER BY NLSSORT(r_reason_desc, 'NLS_SORT =
SCHINESE_PINYIN_M');

-- sorting by case-insensitive order (optional; only English characters are supported in the case-insensitive
order.)
gaussdb=# SELECT * FROM tpcds.reason ORDER BY NLSSORT(r_reason_desc, 'NLS_SORT = generic_m_ci');
```

```
-- Create a partitioned table tpcds.reason_p:
gaussdb=# CREATE TABLE tpcds.reason_p
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
)
PARTITION BY RANGE (r_reason_sk)
(
 partition P_05_BEFORE values less than (05),
 partition P_15 values less than (15),
 partition P_25 values less than (25),
 partition P_35 values less than (35),
 partition P_45_AFTER values less than (MAXVALUE)
)
;

-- Insert data.
gaussdb=# INSERT INTO tpcds.reason_p values(3,'AAAAAAAABAAAAAAA','reason 1'),
(10,'AAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAABAAAAAAA','reason 3'),
(10,'AAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAABAAAAAAA','reason 5'),
(20,'AAAAAAAACAAAAAAA','reason 6'),(30,'AAAAAAAACAAAAAAA','reason 7');

-- Example of the PARTITION clause: Obtain data from the P_05_BEFORE partition in the tpcds.reason_p
table.
gaussdb=# SELECT * FROM tpcds.reason_p PARTITION (P_05_BEFORE);
 r_reason_sk | r_reason_id | r_reason_desc
-----+-----+-----
 3 | AAAAAAAAABAAAAAAA | reason 1
 4 | AAAAAAAAABAAAAAAA | reason 3
(2 rows)

-- Example of the GROUP BY clause: Group records in the tpcds.reason_p table by r_reason_id, and count
the number of records in each group.
gaussdb=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id;
 count | r_reason_id
-----+-----
 2 | AAAAAAAAACAAAAAAA
 5 | AAAAAAAAABAAAAAAA
(2 rows)

-- Example of the GROUP BY CUBE clause: Filter data based on query conditions, and group the results.
gaussdb=# SELECT * FROM tpcds.reason GROUP BY CUBE (r_reason_id,r_reason_sk,r_reason_desc);

-- Example of the GROUP BY GROUPING SETS clause: Filter data based on query conditions, and group the
results.
gaussdb=# SELECT * FROM tpcds.reason GROUP BY GROUPING SETS
((r_reason_id,r_reason_sk),r_reason_desc);

-- Example of the HAVING clause: Group records in the tpcds.reason_p table by r_reason_id, count the
number of records in each group, and display only values whose number of r_reason_id is greater than 2.
gaussdb=# SELECT COUNT(*) c,r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING c>2;
 c | r_reason_id
---+-----
 5 | AAAAAAAAABAAAAAAA
(1 row)

-- Example of the IN clause: Group records in the tpcds.reason_p table by r_reason_id, count the number
of records in each group, and display only the numbers of records whose r_reason_id is
AAAAAAAABAAAAAAA or AAAAAAAADAAAAAAA.
gaussdb=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING r_reason_id
IN('AAAAAAAABAAAAAAA','AAAAAAAADAAAAAAA');
 count | r_reason_id
-----+-----
 5 | AAAAAAAAABAAAAAAA
(1 row)

-- Example of the INTERSECT clause: Query records whose r_reason_id is AAAAAAAABAAAAAAA and
whose r_reason_sk is smaller than 5.
```

```
gaussdb=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAAA' INTERSECT SELECT
* FROM tpcds.reason_p WHERE r_reason_sk<5;
 r_reason_sk | r_reason_id | r_reason_desc
-----+-----+-----
 4 | AAAAAAAAABAAAAAAAA | reason 3
 3 | AAAAAAAAABAAAAAAAA | reason 1
(2 rows)

-- Example of the EXCEPT clause: Query records whose r_reason_id is AAAAAAAAABAAAAAAAA and whose
r_reason_sk is greater than or equal to 4.
gaussdb=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAAA' EXCEPT SELECT *
FROM tpcds.reason_p WHERE r_reason_sk<4;
 r_reason_sk | r_reason_id | r_reason_desc
-----+-----+-----
 10 | AAAAAAAAABAAAAAAAA | reason 5
 10 | AAAAAAAAABAAAAAAAA | reason 4
 4 | AAAAAAAAABAAAAAAAA | reason 3
 10 | AAAAAAAAABAAAAAAAA | reason 2
(4 rows)

-- Create the store_returns and customer tables.
gaussdb=# CREATE TABLE tpcds.store_returns (sr_item_sk int, sr_customer_id varchar(50),sr_customer_sk
int);
gaussdb=# CREATE TABLE tpcds.customer (c_item_sk int, c_customer_id varchar(50),c_customer_sk int);
-- Specify the operator (+) in the WHERE clause to indicate a left join.
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
t1.sr_customer_sk = t2.c_customer_sk(+) ORDER BY 1 DESC LIMIT 1;
 sr_item_sk | c_customer_id
-----+-----
 18000 |
(1 row)

-- Specify the operator (+) in the WHERE clause to indicate a right join.
gaussdb=#
SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
t1.sr_customer_sk(+) = t2.c_customer_sk
ORDER BY 1 DESC LIMIT 1;
 sr_item_sk | c_customer_id
-----+-----
 | AAAAAAAAJNGEBAAA
(1 row)

-- Specify the operator (+) in the WHERE clause to indicate a left join and add a join condition.
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
t1.sr_customer_sk = t2.c_customer_sk(+) AND t2.c_customer_sk(+) < 1 ORDER BY 1 LIMIT 1;
 sr_item_sk | c_customer_id
-----+-----
 1 |
(1 row)

-- If the operator (+) is specified in the WHERE clause, do not use expressions connected through AND or
OR.
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
NOT(t1.sr_customer_sk = t2.c_customer_sk(+) AND t2.c_customer_sk(+) < 1);
ERROR: Operator "(+)" can not be used in nesting expression.
LINE 1: ...tomer_id from store_returns t1, customer t2 where not(t1.sr_...
 ^

-- If the operator (+) is specified in the WHERE clause which does not support expression macros, an error
will be reported.
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
(t1.sr_customer_sk = t2.c_customer_sk(+))::bool;
ERROR: Operator "(+)" can only be used in common expression.

-- If the operator (+) is specified on both sides of an expression in the WHERE clause, an error will be
reported.
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
t1.sr_customer_sk(+) = t2.c_customer_sk(+);
ERROR: Operator "(+)" can't be specified on more than one relation in one join condition
HINT: "t1", "t2"...are specified Operator "(+)" in one condition.
```

```
-- Drop the table.
gaussdb=# DROP TABLE tpceds.reason_p;

-- Example of a WITH RECURSIVE query: Calculate the accumulated value from 1 to 100.
gaussdb=# WITH RECURSIVE t1(a) AS (
 SELECT 100
),
t(n) AS (
 VALUES (1)
 UNION ALL
 SELECT n+1 FROM t WHERE n < (SELECT max(a) FROM t1)
)
SELECT sum(n) FROM t;
sum

5050
(1 row)

-- Example of the UNPIVOT clause: Convert the math and phy columns in the table p1 to the rows (class,
score).
gaussdb=# CREATE TABLE p1(id int, math int, phy int);
gaussdb=# INSERT INTO p1 values(1,20,30);
gaussdb=# INSERT INTO p1 values(2,30,40);
gaussdb=# INSERT INTO p1 values(3,40,50);
gaussdb=# SELECT * FROM p1;
id | math | phy
---+----+---
1 | 20 | 30
2 | 30 | 40
3 | 40 | 50
(3 rows)

gaussdb=# SELECT * FROM p1 UNPIVOT(score FOR class IN(math, phy));
id | class | score
---+----+---
1 | MATH | 20
1 | PHY | 30
2 | MATH | 30
2 | PHY | 40
3 | MATH | 40
3 | PHY | 50
(6 rows)

-- Example of the PIVOT clause: Convert the rows (class, score) in the table p2 to the math and phy
columns.
gaussdb=# CREATE TABLE p2(id int, class varchar(10), score int);
gaussdb=# INSERT INTO p2 SELECT * FROM p1 UNPIVOT(score FOR class IN(math, phy));
gaussdb=# SELECT * FROM p2;
id | class | score
---+----+---
1 | MATH | 20
1 | PHY | 30
2 | MATH | 30
2 | PHY | 40
3 | MATH | 40
3 | PHY | 50
(6 rows)

gaussdb=# SELECT * FROM p2 PIVOT(max(score) FOR class IN ('MATH', 'PHY'));
id | 'MATH' | 'PHY'
---+----+---
1 | 20 | 30
3 | 40 | 50
2 | 30 | 40
(3 rows)

-- Drop the table.
gaussdb=# DROP TABLE p1, p2, tpceds.reason;
```

```
-- Delete a schema.
gaussdb=# DROP SCHEMA tpceds CASCADE;
```

## 7.14.154 SELECT INTO

### Description

Defines a new table based on a query result and inserts data obtained by query to the new table.

Different from SELECT, data is not returned to the client. The table columns have the same names and data types as the output columns of SELECT.

### Precautions

CREATE TABLE AS provides functions similar to SELECT INTO in functions and provides a superset of functions provided by SELECT INTO. You are advised to use CREATE TABLE AS, because SELECT INTO cannot be used in a stored procedure.

### Syntax

```
[WITH [RECURSIVE] with_query [, ...]]
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
 { * | {expression [[AS] output_name]} [, ...] }
 INTO [UNLOGGED] [TABLE] new_table
 [FROM from_item [, ...]]
 [WHERE condition]
 [GROUP BY expression [, ...]]
 [HAVING condition [, ...]]
 [WINDOW {window_name AS (window_definition)} [, ...]]
 [{ UNION | INTERSECT | EXCEPT | MINUS } [ALL | DISTINCT] select]
 [ORDER BY {expression [[ASC | DESC | USING operator] | nlssort_expression_clause] [NULLS { FIRST |
 LAST }]} [, ...]]
 [LIMIT { count | ALL }]
 [OFFSET start [ROW | ROWS]]
 [FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY]
 [{FOR { UPDATE | SHARE } [OF table_name [, ...]] [NOWAIT | WAIT N]} [...]];
```

### Parameters

- **new\_table**  
Specifies the name of the new table.
- **UNLOGGED**  
Specifies that the table is created as an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, an unlogged table is automatically truncated after a crash or unclean shutdown. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well.
  - Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.
  - Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should rebuild the indexes with errors.

 NOTE

For details about other SELECT INTO parameters, see [Parameters](#) in "SELECT."

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAABAAAAAAA','reason 2'),(3,'AAAAAAAABAAAAAAA','reason 3'),
(4,'AAAAAAAABAAAAAAA','reason 4'),(4,'AAAAAAAABAAAAAAA','reason 5'),
(4,'AAAAAAAACAAAAAAA','reason 6'),(5,'AAAAAAAACAAAAAAA','reason 7');

-- Add the values that are less than 5 in the r_reason_sk column in the tpcds.reason table to the new table.
gaussdb=# SELECT * INTO tpcds.reason_t1 FROM tpcds.reason WHERE r_reason_sk < 5;
INSERT 0 6

-- Delete the table.
gaussdb=# DROP TABLE tpcds.reason_t1, tpcds.reason;

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Helpful Links

### [SELECT](#)

## Suggestions

- **DATABASE**  
You are advised not to reindex a database in a transaction.
- **SYSTEM**  
You are advised not to reindex system catalogs in transactions.

## 7.14.155 SET

### Description

Modifies a run-time parameter.

### Precautions

Most run-time parameters can be modified by executing **SET**. Some parameters cannot be modified after a server or session starts.

### Syntax

- Set the system time zone.  
SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT };

- Set the schema of the table.  
SET [ SESSION | LOCAL ]  
{CURRENT\_SCHEMA { TO | = } { schema | DEFAULT }  
| SCHEMA 'schema'};
- Set client encoding.  
SET [ SESSION | LOCAL ] NAMES encoding\_name;
- Set XML parsing mode.  
SET [ SESSION | LOCAL ] XML OPTION { DOCUMENT | CONTENT };
- Set other GUC parameters.  
SET [ LOCAL | SESSION ]  
{config\_parameter { { TO | = } { value | DEFAULT }  
| FROM CURRENT }};

## Parameters

- **SESSION**  
Specifies that the specified parameters take effect for the current session. This is the default value if neither **SESSION** nor **LOCAL** appears.  
If **SET** or **SET SESSION** is executed within a transaction that is later aborted, the effects of the **SET** statement disappear when the transaction is rolled back. Once the surrounding transaction is committed, the effects will persist until the end of the session, unless overridden by another **SET**.
- **LOCAL**  
Specifies that the specified parameters take effect for the current transaction. After **COMMIT** or **ROLLBACK**, the session-level setting takes effect again.  
The effects of this command last only till the end of the current transaction, whether committed or not. A special case is **SET** followed by **SET LOCAL** within a single transaction: the **SET LOCAL** value will be seen until the end of the transaction, but afterward (if the transaction is committed) the **SET** value will take effect.
- **TIME\_ZONE timezone**  
Specifies the local time zone for the current session.  
Value range: a valid local time zone. The corresponding run-time parameter is **TimeZone**. The default value is **PRC**.
- **CURRENT\_SCHEMA schema**  
Specifies the current schema.  
Value range: an existing schema name
- **SCHEMA schema**  
Specifies the current schema. Here the schema is a string.  
Example: set schema 'public';
- **NAMES encoding\_name**  
Specifies the client character encoding. This statement is equivalent to **set client\_encoding to encoding\_name**.  
Value range: a valid character encoding name. The run-time parameter corresponding to this option is **client\_encoding**. The default encoding is **UTF8**.
- **XML OPTION option**  
Specifies the XML parsing mode.

Value range: **CONTENT** (default), **DOCUMENT**

- **config\_parameter**

Specifies the name of a configurable run-time parameter. You can use **SHOW ALL** to view available run-time parameters.

 **NOTE**

Some parameters that viewed by **SHOW ALL** cannot be set by **SET**. For example, **max\_datanodes**.

- **value**

Specifies the new value of **config\_parameter**. This parameter can be specified as string constants, identifiers, numbers, or comma-separated lists of these.

**DEFAULT** can be written to indicate resetting the parameter to its default value.

## Examples

```
-- Set the search path of a schema.
gaussdb=# SET search_path TO tpcds, public;

-- Set the date style to the traditional POSTGRES style (date placed before month):
gaussdb=# SET datestyle TO postgres;
```

## Helpful Links

[RESET](#) and [SHOW](#)

## 7.14.156 SET CONSTRAINTS

### Description

Sets the behavior of constraint checking within the current transaction.

**IMMEDIATE** constraints are checked at the end of each statement. **DEFERRED** constraints are not checked until transaction commit. Each constraint has its own **IMMEDIATE** or **DEFERRED** mode.

Upon creation, a constraint is given one of three characteristics **DEFERRABLE INITIALLY DEFERRED**, **DEFERRABLE INITIALLY IMMEDIATE**, or **NOT DEFERRABLE**. The third class is always **IMMEDIATE** and is not affected by the **SET CONSTRAINTS** statement. The first two classes start every transaction in specified modes, but its behaviors can be changed within a transaction by **SET CONSTRAINTS**.

**SET CONSTRAINTS** with a list of constraint names changes the mode of just those constraints (which must all be deferrable). If multiple constraints match a name, the name is affected by all of these constraints. **SET CONSTRAINTS ALL** changes the modes of all deferrable constraints.

When **SET CONSTRAINTS** changes the mode of a constraint from **DEFERRED** to **IMMEDIATE**, any data that is modified at the end of the transaction is checked during the execution of the **SET CONSTRAINTS** statement. If any such constraint is violated, the **SET CONSTRAINTS** fails (and does not change the constraint mode). Therefore, **SET CONSTRAINTS** can be used to force checking of constraints to occur at a specific point in a transaction. Check and unique constraints are always checked immediately when a row is inserted or modified.

## Precautions

**SET CONSTRAINTS** sets the behavior of constraint checking only within the current transaction. Therefore, if you execute this statement outside of a transaction block (**START TRANSACTION/COMMIT** pair), it will not appear to have any effect.

## Syntax

```
SET CONSTRAINTS { ALL | { name } [, ...] } { DEFERRED | IMMEDIATE } ;
```

## Parameters

- **name**  
Specifies the constraint name.  
Value range: an existing table name, which can be found in the system catalog **pg\_constraint**.
- **ALL**  
Specifies all constraints.
- **DEFERRED**  
Specifies that constraints are not checked until transaction commit.
- **IMMEDIATE**  
Specifies that constraints are checked at the end of each statement.

## Examples

```
-- Set that constraints are checked when a transaction is committed.
gaussdb=# SET CONSTRAINTS ALL DEFERRED;
```

## 7.14.157 SET ROLE

### Description

Sets the current user identifier of the current session.

### Precautions

- Users of the current session must be members of specified **rolename**, but system administrators can choose any roles when separation of duties is disabled.
- Executing this statement may add rights of a user or restrict rights of a user. If the role of a session user has the **INHERITS** attribute, it automatically has all rights of roles that **SET ROLE** enables the role to be. In this case, **SET ROLE** physically deletes all rights directly granted to session users and rights of its belonging roles and only leaves rights of the specified roles. If the role of the session user has the **NOINHERITS** attribute, **SET ROLE** deletes rights directly granted to the session user and obtains rights of the specified role.

### Syntax

- Set the current user identifier of the current session.  

```
SET [SESSION | LOCAL] ROLE role_name PASSWORD 'password';
```

- Reset the current user identifier to that of the current session.  
RESET ROLE;

## Parameters

- **SESSION**  
Specifies that the statement takes effect only for the current session. This parameter is used by default.
- **LOCAL**  
Specifies that the specified statement takes effect only for the current transaction.
- **role\_name**  
Specifies the role name.  
Value range: a string. It must comply with the [naming convention](#).
- **password**  
Specifies the password of a role. It must comply with the password convention.

### NOTE

The restrictions on using a ciphertext password are as follows:

- An administrator cannot use a ciphertext password to switch to another administrator but to a user with lower permissions.  
Ciphertext passwords are usually used in `gs_dump` and `gs_dumpall` export scenarios. In other scenarios, you are advised not to use ciphertext passwords directly.
- **RESET ROLE**  
Resets the current user identifier.

## Examples

```
-- Create a role paul.
gaussdb=# CREATE ROLE paul IDENTIFIED BY '*****';

-- Set the current user to paul.
gaussdb=# SET ROLE paul PASSWORD '*****';

-- View the current session user and the current user.
gaussdb=> SELECT SESSION_USER, CURRENT_USER;

-- Reset the current user.
gaussdb=> RESET ROLE;

-- Delete the user.
gaussdb=# DROP USER paul;
```

## 7.14.158 SET SESSION AUTHORIZATION

### Description

Sets the session user identifier and the current user identifier of the current session to a specified user.

## Precautions

The session identifier can be changed only when the initial session user has the SYSADMIN permission. Otherwise, the system supports the statement only when the authenticated username is specified.

## Syntax

- Set the session user identifier and the current user identifier of the current session.  
`SET [ SESSION | LOCAL ] SESSION AUTHORIZATION role_name PASSWORD 'password';`
- Reset the identifiers of the session and current users to the initially authenticated usernames.  
`{SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT  
| RESET SESSION AUTHORIZATION};`

## Parameters

- **SESSION**  
Specifies that the specified parameters take effect for the current session.
- **LOCAL**  
Specifies that the specified statement takes effect only for the current transaction.
- **role\_name**  
Specifies the username.  
Value range: a string. It must comply with the [naming convention](#).
- **password**  
Specifies the password of a role. It must comply with the password convention.

### NOTE

The restrictions on using a ciphertext password are as follows:

- An administrator cannot use a ciphertext password to switch to another administrator but to a user with lower permissions.  
Ciphertext passwords are usually used in `gs_dump` and `gs_dumpall` export scenarios. In other scenarios, you are advised not to use ciphertext passwords directly.
- **DEFAULT**  
Resets the identifiers of the session and current users to the initially authenticated usernames.

## Examples

```
-- Create a role paul.
gaussdb=# CREATE ROLE paul IDENTIFIED BY '*****';

-- Set the current user to paul.
gaussdb=# SET SESSION AUTHORIZATION paul password '*****';

-- View the current session user and the current user.
gaussdb=> SELECT SESSION_USER, CURRENT_USER;

-- Reset the current user.
gaussdb=> RESET SESSION AUTHORIZATION;
```

```
-- Delete the user.
gaussdb=# DROP USER paul;
```

## Reference

### SET ROLE

## 7.14.159 SET TRANSACTION

### Description

Sets the transaction characteristics. Available transaction characteristics include the transaction separation level and transaction access mode (read/write or read only). You can set the current transaction characteristics using **LOCAL** or the default transaction characteristics of a session using **SESSION**.

### Precautions

The current transaction characteristics must be set in a transaction, that is, **START TRANSACTION** or **BEGIN** must be executed before **SET TRANSACTION** is executed. Otherwise, the setting does not take effect.

### Syntax

Set the isolation level and access mode of the transaction.

```
{ SET [LOCAL | SESSION] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
{ ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
| { READ WRITE | READ ONLY } };
```

### Parameters

- **LOCAL**  
Specifies that the specified statement takes effect only for the current transaction.
- **SESSION**  
Specifies that the specified parameters take effect for the current session.
- **ISOLATION LEVEL**  
Specifies the transaction isolation level that determines the data that a transaction can view if other concurrent transactions exist.

#### NOTE

- The isolation level cannot be changed after data is modified using **INSERT**, **DELETE**, **UPDATE**, **FETCH**, or **COPY** in the current transaction.

Value range:

- **READ COMMITTED**: Only committed data is read. It is the default value.
- **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **SERIALIZABLE**: Currently, this isolation level is not supported in GaussDB. It is equivalent to **REPEATABLE READ**.

- **READ WRITE | READ ONLY**

Specifies the transaction access mode (read/write or read only).

 **NOTE**

The access mode of the default transaction feature of the session can be set only when the database is started or by sending the HUP signal.

## Examples

```
-- Start a transaction and set its isolation level to READ COMMITTED and access mode to READ ONLY.
gaussdb=# START TRANSACTION;
gaussdb=# SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
gaussdb=# COMMIT;
```

## 7.14.160 SHOW

### Description

Shows the current value of a run-time parameter.

### Syntax

```
SHOW
{
 [VARIABLES LIKE] configuration_parameter |
 CURRENT_SCHEMA |
 TIME_ZONE |
 TRANSACTION ISOLATION LEVEL |
 SESSION AUTHORIZATION |
 ALL
};
```

### Parameters

See [Parameters](#) in "RESET."

### Examples

```
-- Show the value of timezone.
gaussdb=# SHOW timezone;

-- Show all parameters.
gaussdb=# SHOW ALL;

-- Show all parameters whose names contain var.
gaussdb=# SHOW VARIABLES LIKE var;
```

### Helpful Links

[SET](#) and [RESET](#)

## 7.14.161 SHUTDOWN

### Description

SHUTDOWN shuts down the currently connected database node.

## Precautions

- Only the administrator can run this command.
- The distributed database does not support the **SHUTDOWN** command.

## Syntax

```
SHUTDOWN [FAST | IMMEDIATE];
```

## Parameters

- ""  
If the shutdown mode is not specified, the default mode **fast** is used.
- **fast**  
Rolls back all active transactions, forcibly disconnects the client, and shuts down the database node without waiting for the client to disconnect.
- **immediate**  
Shuts down the server forcibly. Fault recovery will occur on the next startup.

## Examples

```
-- Shut down the current database node.
gaussdb=# SHUTDOWN;

-- Shut down the current database node in fast mode.
gaussdb=# SHUTDOWN FAST;
```

## 7.14.162 START TRANSACTION

### Description

Starts a transaction. If the isolation level or read/write mode is specified, a new transaction will have those characteristics. You can also specify them using [SET TRANSACTION](#).

### Syntax

Format 1: START TRANSACTION

```
START TRANSACTION
[
 {
 ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
 | { READ WRITE | READ ONLY }
 } [...]
];
```

Format 2: BEGIN

```
BEGIN [WORK | TRANSACTION]
[
 {
 ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
 | { READ WRITE | READ ONLY }
 } [...]
];
```

## Parameters

- **WORK | TRANSACTION**  
Specifies the optional keyword in BEGIN format without functions.
- **ISOLATION LEVEL**  
Specifies the transaction isolation level that determines the data that a transaction can view if other concurrent transactions exist.

### NOTE

The isolation level of a transaction cannot be reset after the first clause (INSERT, DELETE, UPDATE, FETCH, or COPY) for modifying data is executed in the transaction.

Value range:

- **READ COMMITTED**: Only committed data is read. It is the default value.
  - **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
  - **SERIALIZABLE**: Currently, this isolation level is not supported in GaussDB. It is equivalent to **REPEATABLE READ**.
- **READ WRITE | READ ONLY**  
Specifies the transaction access mode (read/write or read only).

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason (c1 int, c2 int);

-- Start a transaction in default mode.
gaussdb=# START TRANSACTION;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# END;

-- Start a transaction in default mode.
gaussdb=# BEGIN;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# END;

-- Start a transaction with the isolation level being READ COMMITTED and the access mode being READ WRITE:
gaussdb=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# COMMIT;

-- Delete the table.
gaussdb=# DROP TABLE tpcds.reason;

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Helpful Links

[COMMIT | END, ROLLBACK](#), and [SET TRANSACTION](#)

## 7.14.163 TIMECAPSULE TABLE

### Description

The **TIMECAPSULE TABLE** statement restores a table to an earlier state in the event of human or application errors.

The table can flash back to a past point in time, depending on the source version of the data stored in the system. In addition, GaussDB cannot restore a table to an earlier state through DDL operations that has changed the structure of the table.

### Precautions

- The **TIMECAPSULE TABLE** statement can be used to flash back the data of the source version or the data from the recycle bin.
  - **TO TIMECAPSULE** and **TO CSN** can flash back a table to an earlier version.
  - The recycle bin records the objects dropped or truncated by running **DROP** and **TRUNCATE**. **TO BEFORE DROP** and **TO BEFORE TRUNCATE** flash back from the recycle bin.
- The following object types do not support flashback: system catalogs, DFS tables, global temporary tables, local temporary tables, unlogged tables, sequence tables, encrypted tables, and hash bucket tables.
- Flashback is not supported for tables that contain user-defined types.
- After flashback is enabled, tables in the recycle bin can be backed up but cannot be restored.
- Between the flashback point and the current point, a statement (DDL, DCL, or VACUUM FULL) that modifies the table structure or affects physical storage has been executed. Therefore, the flashback fails.
- To run **DROP**, you must have the CREATE or USAGE permission on the schema to which the junk object belongs, and you must be the owner of the schema or the owner of the junk object.

To run **TRUNCATE**, you must have the CREATE or USAGE permission on the schema to which the junk object belongs, and you must be the owner of the schema or the junk object. In addition, you must have the TRUNCATE permission on the junk object.

- Scenarios or tables that do not support DROP or TRUNCATE:
  - Scenario where the recycle bin is disabled (**enable\_recyclebin** is set to **off**)
  - Scenario where the system is being maintained (**xc\_maintenance\_mode** is set to **on**) or is being upgraded
  - Scenario where multiple objects are deleted (The **DROP** or **TRUNCATE TABLE** command is executed to delete multiple objects at the same time.)
  - System catalogs, DFS tables, global temporary tables, local temporary tables, UNLOGGED tables, sequence tables, and hash bucket tables.
  - If the object on which the table depends is an external object, the table is physically deleted and is not moved to the recycle bin.

- DROP FLASHBACK constraints:

You can specify either the original user-defined name of the table or the system-generated name assigned to the object when it was dropped.

  - System-generated recycle bin object names are unique. Therefore, if you specify the system-generated name, the database retrieves that specified object. To see the content in your recycle bin, run **select \* from gs\_recyclebin;**
  - If you specify a user-specified name and the recycle bin contains more than one object of that name, the database retrieves the object that was moved to the recycle bin most recently. To retrieve a table of an earlier version, perform the following steps:
    - Specify the system-generated recycle bin name of the table you want to retrieve.
    - Run the **TIMECAPSULE TABLE ... TO BEFORE DROP** statement until you retrieve the table you want.
  - During table restoration, the dropped table along with its original base table name is restored. The subobject names of the table remain the same as those in the recycle bin. You can run the DDL command to manually change the names of subobjects as required.
  - If a table has default values that reference sequences and user-defined functions, flashing back the dropped table will succeed, but the default values will not be restored.
  - Similarly, if a table is referenced by views, dropping the table will cascadingly delete the views. Therefore, flashing back the dropped table will succeed, but the views will not be restored.
  - The recycle bin does not support write operations such as DML, DCL, and DDL, and does not support DQL query operations (supported in later versions).
  - The **recyclebin\_retention\_time** parameter specifies the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires.
  - DROP on multiple tables cannot be restored.
  - Restoration is not supported when accounts or schemas are deleted in cascading mode.
  - When an account or schema is deleted, if the schema or schema object exists in the recycle bin, the common deletion fails and the schema or schema needs to be deleted in cascading mode.
- TRUNCATE FLASHBACK constraints:
  - After TRUNCATE FLASHBACK is performed, the statistics remain unchanged and are still displayed as **0**. You can modify the statistics by manual ANALYZE during off-peak hours (to reduce the impact on performance).
  - RENAME TO supports only the DROP FLASHBACK operation to specify a new name for the retrieval table, but does not support TRUNCATE FLASHBACK.
  - TRUNCATE FLASHBACK cannot span statements that affect the table structure or physical storage. Otherwise, an error is reported. If a

statement (such as DDL, DCL, VACUUM FULL, and partition adding/deleting/splitting/merging) that modifies the table structure or affects physical storages are executed between the flashback point and current point, the flashback fails. The error message "ERROR: The table definition of %s has been changed." is displayed when flashback is performed on a table where DDL operations have been performed. The error message "ERROR: recycle object %s desired does not exist" is displayed when flashback is performed on a table where DDL operations, such as changing namespaces and table names, have been performed.

## Prerequisites

- The **enable\_recyclebin** parameter has been enabled to enable the recycle bin. Contact an administrator for details about how to use the parameter.
- The **recyclebin\_retention\_time** parameter has been set for specifying the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires. Contact an administrator for details about how to use the parameter.

## Syntax

```
TIMECAPSULE TABLE [schema.]table_name TO { CSN expr | TIMESTAMP expr | BEFORE { DROP [RENAME TO table_name] | TRUNCATE } }
```

## Parameters

- **schema**  
Specifies a schema containing the table to be flashed back. If this parameter is not specified, the current schema is used.
- **table\_name**  
Specifies a table name.
- **TO CSN**  
Specifies the CSN corresponding to the time point when the table is to be flashed back. *expr* must be a number representing a valid CSN.  
Note: In the GTM-free scenario, each node uses the local CSN and does not have a globally unified CSN. Therefore, flashback in TO CSN mode is not supported.
- **TO TIMESTAMP**  
Specifies a timestamp value corresponding to the point in time to which you want to flash back the table. The result of *expr* must be a valid past timestamp (convert a string to a time type using the **TO\_TIMESTAMP** function). The table will be flashed back to a time within approximately 3 seconds of the specified timestamp.  
Note: When the flashback point is too old, the source version cannot be obtained because it is recycled. As a result, the flashback fails and the error message "Restore point too old" is displayed.
- **TO BEFORE DROP**  
Retrieves dropped tables and their subobjects from the recycle bin.
- **RENAME TO**  
Specifies a new name for the table retrieved from the recycle bin.

- **TO BEFORE TRUNCATE**

Flashes back to the point in time before the TRUNCATE operation.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Delete the tpcds.reason_t2 table.
DROP TABLE IF EXISTS tpcds.reason_t2;
-- Create the tpcds.reason_t2 table.
gaussdb=# CREATE TABLE tpcds.reason_t2
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
)with(storage_type = ustore);
-- Insert records into the tpcds.reason_t2 table.
gaussdb=# INSERT INTO tpcds.reason_t2 VALUES (1, 'AA', 'reason1'),(2, 'AB', 'reason2'),(3, 'AC', 'reason3');
INSERT 0 3
-- Delete data from the tpcds.reason_t2 table.
gaussdb=# TRUNCATE TABLE tpcds.reason_t2;
-- Query data in the tpcds.reason_t2 table.
gaussdb=# select * from tpcds.reason_t2;
 r_reason_sk | r_reason_id | r_reason_desc
-----+-----+-----
(0 rows)
-- Perform the TRUNCATE FLASHBACK operation.
gaussdb=# TIMECAPSULE TABLE tpcds.reason_t2 to BEFORE TRUNCATE;
gaussdb=# select * from tpcds.reason_t2;
 r_reason_sk | r_reason_id | r_reason_desc
-----+-----+-----
+-----+-----+-----
 1 | AA | reason1
 2 | AB | reason2
 3 | AC | reason3
(3 rows)
-- Delete the tpcds.reason_t2 table.
gaussdb=# DROP TABLE tpcds.reason_t2;
-- Perform the DROP FLASHBACK operation.
gaussdb=# TIMECAPSULE TABLE tpcds.reason_t2 to BEFORE DROP;
TimeCapsule Table
-- Perform DROP Flashback to specify a new name for the retrieval table.
gaussdb=# TIMECAPSULE TABLE tpcds.reason_t2 TO BEFORE DROP rename to reason_t3;
TimeCapsule Table
-- Clear the recycle bin and delete the schema.
gaussdb=# PURGE RECYCLEBIN;
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## 7.14.164 TRUNCATE

### Description

TRUNCATE quickly removes all rows from a database table.

It has the same effect as an unqualified DELETE on each table, but TRUNCATE is faster because it does not actually scan the tables. This is most useful on large tables.

### Precautions

- TRUNCATE TABLE has the same function as a DELETE statement with no WHERE clause, emptying a table.

- TRUNCATE TABLE uses less system and transaction log resources as compared with DELETE.
  - DELETE deletes a row each time, and records the deletion of each row in the transaction log.
  - TRUNCATE TABLE deletes all rows in a table by releasing the data page storing the table data, and records the releasing of the data page only in the transaction log.
- The differences between TRUNCATE, DELETE, and DROP are as follows:
  - TRUNCATE TABLE deletes content and releases space, but does not delete definitions.
  - DELETE TABLE deletes content, but does not delete definitions nor release space.
  - DROP TABLE deletes content and definitions, and releases space.

## Syntax

- Delete data from a table.

```
TRUNCATE [TABLE] [ONLY] { table_name [*] } [, ...]
[CONTINUE IDENTITY] [CASCADE | RESTRICT] [PURGE];
```

- Truncate the data in a partition.

```
ALTER TABLE [IF EXISTS] { [ONLY] table_name
| table_name *
| ONLY (table_name) }
TRUNCATE PARTITION { partition_name
| FOR (partition_value [, ...]) } [UPDATE GLOBAL INDEX];
```

## Parameters

- **ONLY**  
If **ONLY** is specified, only the specified table is cleared. Otherwise, the table and all its subtables (if any) are cleared.
- **table\_name**  
Specifies the name (optionally schema-qualified) of the target table.  
Value range: an existing table name
- **CONTINUE IDENTITY**  
Does not change the values of sequences. This is the default action.
- **CASCADE | RESTRICT**
  - **CASCADE**: Clears all tables that are added to a group.
  - **RESTRICT** (default): refuses to truncate if any of the tables have foreign-key references from tables that are not listed in the statement (not supported in distributed scenarios).
- **PURGE**  
Purges table data in the recycle bin by default.
- **partition\_name**  
Specifies the partition in the target partitioned table.  
Value range: an existing table name.
- **partition\_value**  
Specifies the value of the specified partition key.

You can use PARTITION FOR to uniquely identify a partition.

Value range: value range of the partition key for the partition to be renamed

#### NOTICE

When the PARTITION FOR clause is used, the entire partition where **partition\_value** is located is cleared.

- **UPDATE GLOBAL INDEX**

If this parameter is used, all global indexes in a partitioned table are updated to ensure that correct data can be queried using global indexes.

If this parameter is not used, all global indexes in a partitioned table will become invalid.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.reason values(1,'AAAAAAAAABAAAAAAA','reason 1'),
(5,'AAAAAAAAABAAAAAAA','reason 2'),(15,'AAAAAAAAABAAAAAAA','reason 3'),
(25,'AAAAAAAAABAAAAAAA','reason 4'),(35,'AAAAAAAAABAAAAAAA','reason 5'),
(45,'AAAAAAAAACAAAAAAA','reason 6'),(55,'AAAAAAAAACAAAAAAA','reason 7');

-- Create a table.
gaussdb=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

-- Clear the tpcds.reason_t1 table.
gaussdb=# TRUNCATE TABLE tpcds.reason_t1;

-- Delete the table.
gaussdb=# DROP TABLE tpcds.reason_t1;
-- Create a partitioned table.
gaussdb=# CREATE TABLE tpcds.reason_p
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)
(
 partition p_05_before values less than (05),
 partition p_15 values less than (15),
 partition p_25 values less than (25),
 partition p_35 values less than (35),
 partition p_45_after values less than (MAXVALUE)
);

-- Insert data.
gaussdb=# INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;

-- Clear the p_05_before partition.
gaussdb=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;

-- Clear the p_15 partition.
```

```
gaussdb=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (13);
-- Clear the partitioned table.
gaussdb=# TRUNCATE TABLE tpcds.reason_p;

-- Delete the tpcds.reason_p table.
gaussdb=# DROP TABLE tpcds.reason_p;

-- Delete the tpcds.reason table.
gaussdb=# DROP TABLE tpcds.reason;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## 7.14.165 UPDATE

### Description

Updates data in a table. UPDATE changes the values of the specified columns in all rows that satisfy the condition. The WHERE clause clarifies conditions. The columns to be modified need to be mentioned in the SET clause; columns not explicitly modified retain their previous values.

### Precautions

- The owner of a table, users granted with the UPDATE permission on the table, or users granted with the UPDATE ANY TABLE permission can update data in the table. System administrators have the permission to update data in the table by default.
- UPDATE..... LIMIT row\_count supports only some scenarios of execution plan pushdown. (Table replication is not supported.) The prerequisites are that the filter criteria must contain the equivalent distribution key and the filter criteria should be relatively simple. Do not use forcible type conversion. If the command fails to be executed, simplify the filter criteria.
- You must have the SELECT permission on all tables involved in the expressions or conditions.
- The distribution key (or column) of a table cannot be modified.
- For the UPDATE statement whose subquery is a stream plan, the same row cannot be concurrently updated.
- You are not allowed to change the database encoding format to GB18030\_2022 or change GB18030\_2022 to another character encoding format by updating system catalogs. Otherwise, inventory data and some operations will be abnormal. If you need to change the character set encoding of a database, follow the database switching process to migrate data.

### Syntax

```
[WITH [RECURSIVE] with_query [, ...]]
UPDATE [/*+ plan_hint */] [ONLY] table_name [*] [[AS] alias]
SET {column_name = { expression | DEFAULT }
 | (column_name [, ...]) = {({ expression | DEFAULT } [, ...]) |sub_query }} [, ...]
[FROM from_list] [WHERE condition | WHERE CURRENT OF cursor_name]
[ORDER BY {expression [[ASC | DESC | USING operator] [LIMIT row_count]
| RETURNING {*
 | {output_expression [[AS] output_name } [, ...] }}];
```

where sub\_query can be:

```
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
{ * | {expression [[AS] output_name]} [, ...] }
[FROM from_item [, ...]]
[WHERE condition | WHERE CURRENT OF cursor_name]
[GROUP BY grouping_element [, ...]]
[HAVING condition [, ...]]
[ORDER BY {expression [[ASC | DESC | USING operator] | nlssort_expression_clause] [NULLS { FIRST |
LAST }]} [, ...]]
[LIMIT { [offset,] count | ALL }]
```

## Parameters

- **WITH [ RECURSIVE ] with\_query [, ...]**

Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table. This subquery statement structure is called the common table expression (CTE) structure. When this structure is used, the execution plan contains the CTE SCAN content.

If **RECURSIVE** is specified, it allows a **SELECT** subquery to reference itself by name.

The detailed format for **with\_query** is as follows:

```
with_query_name [(column_name [, ...])] AS [[NOT] MATERIALIZED] ({select | values | insert |
update | delete})
```

**with\_query\_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.

- **column\_name** specifies the column name displayed in the subquery result set.
- Each subquery can be a **SELECT**, **VALUES**, **INSERT**, **UPDATE** or **DELETE** statement.
- You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE.
  - If **MATERIALIZED** is specified, the **WITH** query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the **WITH** subquery cannot be jointly optimized with the **SELECT** statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the **WITH** query can be executed as a subquery inline, the preceding optimization can be performed.
  - If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the **SELECT** statement trunk to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

- **plan\_hint** clause

Follows the **UPDATE** keyword in the */\*+ \*/* format. It is used to optimize the plan of an **UPDATE** statement block. For details, see [Hint-based Tuning](#). In each statement, only the first */\*+ plan\_hint \*/* comment block takes effect as a hint. Multiple hints can be written.

- **table\_name**

Specifies the name (optionally schema-qualified) of the table to be updated.

Value range: an existing table name

 NOTE

You can use database links to perform operations on remote tables. For details, see [DATABASE LINK](#).

- **alias**  
Specifies a substitute name for the target table.  
Value range: a string. It must comply with the [naming convention](#).
- **column\_name**  
Specifies the name of the column to be modified.  
You can refer to this column by specifying the target table alias and the column name. Example:  
`UPDATE foo AS f SET f.col_name = namecol';`  
Value range: an existing column
- **expression**  
Specifies a value assigned to a column or an expression that assigns the value.
- **DEFAULT**  
Specifies the default value of a column.  
The value is **NULL** if no specified default value has been assigned to it.
- **sub\_query**  
Specifies a subquery.  
This statement can be executed to update a table with information for other tables in the same database. For details about clauses in the SELECT statement, see [SELECT](#).  
When a single column is updated, the ORDER BY and LIMIT clauses can be used. When multiple columns are updated, the ORDER BY and LIMIT clauses cannot be used.

 NOTE

For an UPDATE statement in `UPDATE t1 SET (c1,c2) = (SELECT c1, c2 FROM t2...)` format, a subplan is generated for each column in the execution plan. When a large number of columns are updated, the number of subplans is large, which greatly affects the performance.

- **from\_list**  
Specifies a list of table expressions. You can use columns of other tables in the WHERE condition. This is similar to the list of tables that can be specified in the FROM clause of a SELECT statement.

---

**NOTICE**

Note that the target table cannot appear in **from\_list**, unless you intend a self-join (in which case it must appear with an alias in **from\_list**).

- **condition**  
Specifies an expression that returns a value of type Boolean. Only rows for which this expression returns **true** are updated. You are advised not to use

numeric types such as int for **condition**, because such types can be implicitly converted to Boolean values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

- **WHERE CURRENT OF cursor\_name**

When the cursor points to a row in a table, you can use this syntax to update the row.

cursor\_name: specifies the name of a cursor.

---

**NOTICE**

- This syntax is not supported by the database in the MySQL compatibility mode.
- This syntax supports only ordinary tables and does not support partitioned tables or hash bucket tables.
- This syntax can be used only in stored procedures.
- This syntax cannot be used together with other WHERE conditions.
- This syntax cannot be used together with WITH, USING, ORDER BY, or FROM.
- The SELECT statement corresponding to the CURSOR must be declared as FOR UPDATE.
- The SELECT statement corresponding to the CURSOR supports only a single table. It does not support LIMIT/OFFSET, subqueries, or sublinks.
- The CURSOR declared as FOR UPDATE in a stored procedure cannot be used after being committed or rolled back.
- If the row to which the CURSOR points does not exist, an error is reported (only when UPDATE is used instead of DELETE) in the ORA-compatible mode, indicating that the specified row does not exist. In other compatibility modes, no error is reported.
- When this syntax is used in a replication table, pushdown is not supported. A primary key is required.

---

- **ORDER BY**

For details about the keywords, see [SELECT](#).

- **LIMIT**

For details about the keywords, see [SELECT](#).

- **output\_expression**

Specifies an expression to be computed and returned by the **UPDATE** statement after each row is updated.

Value range: any table and table columns listed in FROM. Write \* to return all columns.

- **output\_name**

Specifies a name to use for a returned column.

## Examples

```
-- Create the student1 table.
gaussdb=# CREATE TABLE student1
```

```
gaussdb=# (
gaussdb=# stuno int,
gaussdb=# classno int
gaussdb=#)
gaussdb=# DISTRIBUTE BY hash(stuno);

-- Insert data.
gaussdb=# INSERT INTO student1 VALUES(1,1);
gaussdb=# INSERT INTO student1 VALUES(2,2);
gaussdb=# INSERT INTO student1 VALUES(3,3);

-- View data.
gaussdb=# SELECT * FROM student1;

-- Update the values of all records.
gaussdb=# UPDATE student1 SET classno = classno*2;

-- View data.
gaussdb=# SELECT * FROM student1;

-- Delete the table.
gaussdb=# DROP TABLE student1;

-- Example for WHERE CURRENT OF cursor_name
gaussdb=# CREATE TABLE t1(c1 int, c2 varchar2); -- Create a table.
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,1000),'abcd'); -- Insert data.

gaussdb=# DECLARE
gaussdb=# CURSOR cur1 IS SELECT * FROM t1 WHERE c1 = 1 FOR UPDATE;
gaussdb=# va t1%rowtype;
gaussdb=# BEGIN
gaussdb$$$ OPEN cur1;
gaussdb$$$ FETCH cur1 INTO va;
gaussdb$$$ SET t1 SET c2 = c2 || c2 WHERE CURRENT OF cur1; -- Use WHERE CURRENT OF cursor_name
to update data.
gaussdb$$$ CLOSE cur1;
gaussdb$$$ COMMIT;
gaussdb$$$ END;
gaussdb$$$ /

gaussdb=# SELECT * FROM t1 WHERE c1 = 1; -- Query data.
```

## 7.14.166 VACUUM

### Description

VACUUM recycles storage space occupied by deleted rows in tables or B-Tree indexes. In normal database operation, rows that have been deleted are not physically removed from their table; they remain present until a VACUUM is done. Therefore, it is necessary to execute VACUUM periodically, especially on frequently-updated tables.

### Precautions

- If no table is specified, VACUUM processes the tables on which the user has the corresponding permission in the current database. With a parameter, VACUUM processes only that table.
- To perform VACUUM operation to a table, you must be a table owner or a user granted the VACUUM permission on the table. System administrators have this permission by default. However, database owners are allowed to VACUUM all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true database-wide VACUUM can only be

executed by system administrators). VACUUM skips over any tables that the calling user does not have permission to vacuum.

- VACUUM cannot be executed inside a transaction block.
- It is recommended that active production databases be vacuumed frequently (at least nightly), in order to remove dead rows. After adding or deleting a large number of rows, it might be a good idea to run **VACUUM ANALYZE** for the affected table. This will update the system catalogs with the results of all recent changes, and allow the query planner to make better choices in planning queries.
- **FULL** is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table. VACUUM FULL usually shrinks a table more than VACUUM does. The **FULL** option does not clear indexes. You are advised to periodically run the **REINDEX** statement. If the physical space usage does not decrease after you run the statement, check whether there are other active transactions (that have started before you delete data transactions and not ended before you run **VACUUM FULL**). If there are such transactions, run this statement again when the transactions quit.
- **VACUUM** causes a substantial increase in I/O traffic, which might cause poor performance for other active sessions. Therefore, it is sometimes advisable to use the cost-based VACUUM delay feature.
- When **VERBOSE** is specified, **VACUUM** prints progress messages to indicate which table is currently being processed. Various statistics about the tables are printed as well.
- When the option list is surrounded by parentheses, the options can be written in any order. If there are no brackets, the options must be given in the order displayed in the syntax.
- VACUUM and VACUUM FULL clean up row-store table records with a delay specified by **vacuum\_defer\_cleanup\_age**, that is, tuples that have just been deleted are not cleaned up immediately.
- VACUUM ANALYZE executes a VACUUM operation and then an ANALYZE operation for each selected table. This is a handy combination form for routine maintenance scripts.
- A plain VACUUM statement (without the FULL option) simply recycles space and makes it available for reuse. This form of statement can operate in parallel with normal reading and writing of the table, as an exclusive lock is not obtained. VACUUM FULL executes wider processing, including moving rows across blocks to compress the tables so that they occupy the minimum number of disk blocks. This form is much slower and requires an exclusive lock on each table while it is being processed.
- If the **xc\_maintenance\_mode** parameter is not enabled, VACUUM FULL skips all system catalogs.
- If you run **VACUUM FULL** immediately after running **DELETE**, the space will not be reclaimed. After executing **DELETE**, execute 1000 non-SELECT transactions, or wait for 1s and then execute one transaction. Then, run **VACUUM FULL** to reclaim the space.
- During VACUUM FULL, an exclusive lock is added to the table. Therefore, you are advised not to run **VACUUM FULL** during peak hours. Otherwise, the lock waiting time is too long or a deadlock occurs.

- To ensure performance and statistics accuracy, do not run ANALYZE-related commands, such as VACUUM ANALYZE, AUTOANALYZE, and manual ANALYZE, at the same time or frequently.

## Syntax

- Reclaim space and update statistics information, no requirements for keyword orders.

```
VACUUM [({ FULL | FREEZE | VERBOSE | {ANALYZE | ANALYSE } } [,...])]
[table_name [(column_name [, ...])] [PARTITION (partition_name)]];
```

- Reclaim space, without updating statistics information.

```
VACUUM [FULL [COMPACT]] [FREEZE] [VERBOSE] [table_name [PARTITION
(partition_name)]];
```

- Reclaim space and update statistics information, and require keywords in order.

```
VACUUM [FULL] [FREEZE] [VERBOSE] { ANALYZE | ANALYSE } [VERBOSE]
[table_name [(column_name [, ...])] [PARTITION (partition_name)]];
```

## Parameters

- **FULL**

Selects "FULL" vacuum, which can reclaim more space, but takes much longer and exclusively locks the table.

### NOTE

Using FULL will cause statistics information missing. To collect statistics information, add the keyword ANALYZE to the VACUUM FULL statement.

- **FREEZE**

Is equivalent to running **VACUUM** with the **vacuum\_freeze\_min\_age** parameter set to **zero**.

- **VERBOSE**

Prints a detailed vacuum activity report for each table.

- **ANALYZE | ANALYSE**

Updates statistics used by the planner to determine the most efficient way to execute a query.

- **table\_name**

Specifies the name (optionally schema-qualified) of a specific table to vacuum.

Value range: name of a specific table to vacuum Defaults are all tables in the current database.

- **column\_name**

Specifies the name of the column to be analyzed. This parameter must be used together with **ANALYZE**.

Value range: name of a specific column to analyze The default value indicates all columns.

 NOTE

The mechanism of the VACUUM ANALYZE statement is to execute VACUUM and ANALYZE in sequence. Therefore, if **column\_name** is incorrect, VACUUM may be successfully executed but ANALYZE may fail to be executed. For a partitioned table, ANALYZE may fail to be executed after VACUUM is successfully executed on a partition.

- **PARTITION**  
**COMPACT** and **PARTITION** cannot be used at the same time.
- **partition\_name**  
Specifies the partition name of the table to be cleared. The default value indicates all partitions.

## Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.reason values(1,'AAAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAAABAAAAAAA','reason 2');

-- Create an index in the tpcds.reason table.
gaussdb=# CREATE UNIQUE INDEX ds_reason_index1 ON tpcds.reason(r_reason_sk);

-- Perform VACUUM on the tpcds.reason table that has indexes.
gaussdb=# VACUUM (VERBOSE, ANALYZE) tpcds.reason;

-- Delete the index.
gaussdb=# DROP INDEX ds_reason_index1 CASCADE;
gaussdb=# DROP TABLE tpcds.reason;
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

## Suggestions

- VACUUM
  - VACUUM cannot be executed inside a transaction block.
  - It is recommended that active production databases be vacuumed frequently (at least nightly), in order to remove dead rows. It is strongly recommended that you run **VACUUM ANALYZE** after adding or deleting a large number of records.
  - **FULL** is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table.

## 7.14.167 VALUES

### Description

Computes a row or a set of rows based on given values. It is most commonly used to generate a constant table within a large statement.

## Precautions

- VALUES lists with large numbers of rows should be avoided, as you might encounter out-of-memory failures or poor performance. **VALUES** appearing within **INSERT** is a special case, because the desired column types are known from the **INSERT**'s target table, and need not be inferred by scanning the **VALUES** list. In this case, **VALUE** can handle larger lists than are practical in other contexts.
- If more than one row is specified, all the rows must have the same number of elements.

## Syntax

```
VALUES {(expression [, ...])} [, ...]
 [ORDER BY { sort_expression [ASC | DESC | USING operator] } [, ...]]
 [LIMIT { count | ALL }]
 [OFFSET start [ROW | ROWS]]
 [FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY];
```

## Parameters

- **expression**  
Specifies a constant or expression to compute and insert at the indicated place in the resulting table or set of rows.  
In a **VALUES** list appearing at the top level of an **INSERT**, an expression can be replaced by **DEFAULT** to indicate that the destination column's default value should be inserted. **DEFAULT** cannot be used when **VALUES** appears in other contexts.
- **sort\_expression**  
Specifies an expression or integer constant indicating how to sort the result rows.
- **ASC**  
Specifies an ascending sort order.
- **DESC**  
Specifies a descending sort order.
- **operator**  
Specifies a sorting operator.
- **count**  
Specifies the maximum number of rows to return.
- **OFFSET start [ ROW | ROWS ]**  
Specifies the maximum number of returned rows, whereas **start** specifies the number of rows to skip before starting to return rows.
- **FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY**  
The **FETCH** clause restricts the total number of rows starting from the first row of the return query result, and the default value of **count** is **1**.

## Examples

See [Examples](#) in "INSERT."

## 7.15 Appendix

### 7.15.1 Extended Functions

The following table lists the extended functions supported by GaussDB and they are for reference only.

| Type             | Name                                        | Description                                                     |
|------------------|---------------------------------------------|-----------------------------------------------------------------|
| Trigger function | pg_get_triggerdef(trigger_oid)              | Gets <b>CREATE [ CONSTRAINT ] TRIGGER</b> command for triggers. |
|                  | pg_get_triggerdef(trigger_oid, pretty_bool) | Gets <b>CREATE [ CONSTRAINT ] TRIGGER</b> command for triggers. |

### 7.15.2 Dollar-Quoted String Constants

If a string sequence contains a single quotation mark ('), double the single quotation mark (') to two single quotation marks ("). Otherwise, the SQL statement may fail to be executed.

If a string contains many single quotation marks (') or backslashes (\), the string may be difficult to understand and error-prone because all the single quotation marks (') are doubled.

To make the query more readable in such situations, dollar quoting is provided to write string constants. A dollar-quoted string constant consists of a dollar sign (\$), a "tag" of zero or more characters, another dollar sign, an arbitrary sequence of characters that make up the string content, a dollar sign, another tag the same as the previous one, and a dollar sign.

```
gaussdb=# SELECT $$it's an example$$;
?column?

it's an example
(1 row)
```

### 7.15.3 DATABASE LINK

#### Description

In the local database, a database link is used to connect to and access the remote database.

A database link can be public or private. A private database link can be accessed only by the creator, while a public database link can be accessed by all users.

All created database link information is stored in the system view **gs\_db\_links** of the local database.

## Precautions

- The database link feature can be used only in ORA-compatible versions.
- The remote database connected to database link supports only 503.1 and later versions.
- Ensure that the values of the compatibility parameter **DBCOMPATIBILITY** and GUC parameters **behavior\_compat\_options**, **a\_format\_dev\_version**, and **a\_format\_version** of the local and remote databases are the same.

- When a session is enabled for a database link connection, the following GUC parameters are set:

```
set search_path=pg_catalog, '$user', 'public';
set datestyle=ISO;
set intervalstyle=postgres;
set extra_float_digits=3;
```

Other parameters are set at the remote end. If the remote parameters are different from the local parameters, the data display formats may be different. Therefore, ensure that the remote parameters are the same as the local parameters.

- Preparations: Use `gs_guc` to add a whitelist to the **pg\_hba.conf** file to allow client connections.

```
Example: gs_guc reload -I all -N all -Z coordinator -Z datanode -h "host all all 192.168.11.11/32 sha256"
```

For details about the parameters, see the description of `gs_guc` client authentication policy settings.

- In some cases, the IP address of the DN also needs to be added to the cluster whitelist.
- The permission to create a database link needs to be granted using the GRANT syntax. By default, a new user does not have the permission, but system administrators have the permission. For details, see [GRANT](#).
- When a database link is used to perform operations on a remote table, a schema corresponding to the remote table is created locally. If the metadata of the table does not exist locally, the metadata is written to the local system catalog. In this case, a level-7 lock is used to ensure write consistency until the transaction ends. When a database link is deleted, the corresponding metadata is also deleted.
- When DATABASE LINK is used, locally created tables are used only to store metadata of remote tables. The table structure cannot be queried using the `\d` or `pg_get_tabledef` function.
- If a long transaction uses the dblink to operate a remote object for the first time, the lock is held until the transaction ends. Other transactions that use the dblink for the first time are blocked. To avoid this problem, run a quick statement, for example, "select \* from t1@dblink where 1=2;", to query the remote object to be used and flush its metadata to disks. In addition, similar problem also occurs when the structure of the remote table changes and the stored metadata is updated locally.
- When a schema corresponding to the remote end is created locally, "USERNAME (available only for private database link) #remote schema@DBLINK" is used as the schema name. The maximum length of the schema name is 63 characters.
- If the local and remote character sets are different, an error indicating that the conversion fails may be reported. The error information is that the remote

end returns an error. If the character encoding of the local database is GB18030\_2022, the character encoding sent to the remote database is converted to GB18030. Therefore, if the character set of the local database is GB18030\_2022, the character set of the remote database can only be GB18030 or GB18030\_2022.

- If a key file is regenerated after a database link is created, an error is reported when the database link is used. This is because the key file used for creating the database link is different from that used for decryption.
- When a database link is used to perform operations on a remote table, a single-node group is created and randomly bound to a DN.

---

#### NOTICE

When the permission to create a database link is granted to a user, the user can remotely access a database by using the IP address of the remote database. Exercise caution when granting this permission to users.

---

## Syntax

- Create a database link.  

```
CREATE [PUBLIC] DATABASE LINK dblink
[CONNECT TO { CURRENT_USER | user IDENTIFIED BY password }] [USING (option 'value' [...])] ;
```
- Modify the database link information.  

```
ALTER [PUBLIC] DATABASE LINK dblink
{ CONNECT TO user IDENTIFIED BY password } ;
```
- Drop a specified database link.  

```
DROP [PUBLIC] DATABASE LINK dblink;
```

 NOTE

- **PUBLIC**: creates a public database link visible to all users. If this clause is omitted, the database link is private and available only to the current user. The data that can be accessed on the remote database depends on the identity used by the database link during connection.
- If **CONNECT TO user IDENTIFIED BY password** is specified, the database link makes a connection as a user with specified password.
- If **CONNECT TO CURRENT\_USER** is specified, the database link uses the initial username and empty password of the current database to connect to the remote database.
- If the preceding two clauses are omitted, the database connection connects to the remote database as a local initial user.
- **dblink**: indicates the name of the database link to be created.
- **user**: indicates the username used by the created database link.
- **password**: indicates the password of the username.
- **USING ( option 'value' [, ... ] )**

Specifies parameters such as the IP address, port number, and remote database name of the database to be connected. The supported options are as follows:

- **host**: specifies the IP addresses to be connected. IPv6 addresses are not supported. Multiple IP addresses can be specified using character strings separated by commas (.). Currently, SSL settings, encrypted databases, and certificate authentication are not supported. If no IP address is specified, this parameter is left empty by default.
- **port**: specifies the port number for connection. If this parameter is not specified, the default value **5432** is used.
- **dbname**: specifies the name of the database to be connected. If this parameter is not specified, the username used for connecting to the remote end is used by default.
- **fetch\_size**: specifies the amount of data obtained from the remote end each time. The value of **fetch\_size** ranges from 0 to 2147483647. The default value is **100**.

**Notes:**

- You can write only part of the preceding options in the brackets after USING.
- If the keyword USING is not written, the content in the brackets is not written as well.
- When a database link is created, the system does not check whether the connection is successful. If related keywords are missing, an error may be reported.

Perform the SELECT operation through a database link.

 NOTE

The syntax for accessing a remote database object by using a created database link is basically the same as that for accessing a local object. The difference is that **@dblink** is added to the end of the remote object descriptor. For details about restrictions on SQL statements, see [Table 7-168](#).

```
[WITH [RECURSIVE] with_query [, ...]]
SELECT [/*+ plan_hint */] [ALL | DISTINCT [ON (expression [, ...])]]
{ * | {expression [[AS] output_name]} [, ...] }
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY grouping_element [, ...]]
[HAVING condition [, ...]]
[{ UNION | INTERSECT | EXCEPT | MINUS } [ALL | DISTINCT] select]
[ORDER BY {expression [[ASC | DESC | USING operator]] nlsort_expression_clause } [NULLS
```

```
{ FIRST | LAST } } [, ...]
[LIMIT { [offset,] count | ALL }]
[OFFSET start [ROW | ROWS]]
{ FOR { UPDATE | SHARE } [OF table_name [, ...] } [...] };
{ [ONLY] table_name [*] @ dblink [[AS] alias [(column_alias [, ...])]]
| (select) [AS] alias [(column_alias [, ...])]
| with_query_name [[AS] alias [(column_alias [, ...])]]
| [function_name] ([argument [, ...]]) [AS] alias [(column_alias [, ...] | column_definition [, ...])]
| [function_name] ([argument [, ...]]) AS (column_definition [, ...])
| from_item [NATURAL] join_type from_item [ON join_condition | USING (join_column [, ...])]};
```

- Perform the INSERT operation through a database link.

```
[WITH [RECURSIVE] with_query [, ...]]
INSERT [/*+ plan_hint */] INTO table_name @ dblink [(column_name [, ...])]
{ DEFAULT VALUES
| VALUES { ({ expression | DEFAULT } [, ...]) } [, ...]
| query }
[RETURNING { {output_expression [[AS] output_name] } [, ...] }];
```

- Perform the UPDATE operation through a database link.

```
UPDATE [/*+ plan_hint */] [ONLY] table_name @ dblink [[AS] alias]
SET {column_name = { expression | DEFAULT }
| (column_name [, ...]) = { ({ expression | DEFAULT } [, ...]) |sub_query }} [, ...]
[FROM from_list] [WHERE condition]
[ORDER BY {expression [[ASC | DESC | USING operator] | nlssort_expression_clause] [NULLS
{ FIRST | LAST } }] [, ...]]
[LIMIT { [offset,] count | ALL }]
[RETURNING { {output_expression [[AS] output_name] } [, ...] }];
```

where sub\_query can be:

```
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
{ * | {expression [[AS] output_name] } [, ...] }
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY grouping_element [, ...]]
[HAVING condition [, ...]];
```

- Perform the DELETE operation through a database link.

```
[WITH [RECURSIVE] with_query [, ...]]
DELETE [/*+ plan_hint */] FROM [ONLY] table_name @ dblink [[AS] alias]
[USING using_list]
[WHERE condition]
[ORDER BY {expression [[ASC | DESC | USING operator] | nlssort_expression_clause] [NULLS
{ FIRST | LAST } }] [, ...]]
[LIMIT { [offset,] count | ALL }]
[RETURNING { { output_expr [[AS] output_name] } [, ...] }];
```

- Perform the LOCK TABLE operation through a database link.

```
LOCK [TABLE] { [ONLY] name @ dblink [, ...] }
[IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE |
SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE]
[NOWAIT];
```

- Call a stored procedure or function of the remote database.

```
CALL | SELECT [schema.] { func_name@dblink | procedure_name@dblink } (param_expr);
```

 **CAUTION**

- When a database link calls a remote function or stored procedure, the **out** output parameters, aggregate functions, window functions, and return set functions are not supported. `SELECT * FROM func@dblink()` cannot be used to call the function or stored procedure.
- When a database link calls a remote function or stored procedure, functions under PUBLIC are called by default if no schema is specified.
- When the database link calls a remote function or stored procedure, **param\_expr** does not support the use of ":@" or "=>" to separate parameter names and parameter values.

 **NOTE**

- The meanings of the parameters irrelevant to database link in the preceding SQL statements are the same as those in the original SQL statements.
- When specifying a column name, you can add "@dblink" to the end of the column name to specify the column of the table to which the corresponding database link points.
- When a database link is used to execute an UPDATE or DELETE statement with LIMIT, the statement can be executed regardless of whether the WHERE condition is a distribution key.

## Restrictions

- **Transaction**

When a database link is used, the relationship between local and remote transactions is as follows:

- A local transaction synchronously controls the commit/rollback status of a remote transaction.
- The relationship between isolation levels is as follows.

| Local Isolation Level | Remote Isolation Level |
|-----------------------|------------------------|
| Read Uncommitted      | Repeatable Read        |
| Read Committed        | Repeatable Read        |
| Repeatable Read       | Repeatable Read        |
| Serializable          | Serializable           |

**NOTICE**

If you commit a local transaction, a transaction commit request is sent to the remote end. If the local transaction fails to be committed due to an exception (such as a connection exception or a local cluster instance exception) after the remote transaction is successfully committed, the committed remote transaction cannot be withdrawn. As a result, the local transaction may be inconsistent with the remote transaction.

- **Permissions of local users to use database links**
  - a. If the keyword **public** is used, the link is a public database link and can be used by all users or schemas.
  - b. If the keyword **public** is not used, the link is a private database link and can be used only by the current user or schema. (SYSADMIN cannot use database links across schemas.)
- **Permission to access remote database objects through database links**

The permissions to access remote database objects are the same as those of the remote connection user bound to the database link.
- **Supported SQL statements**
  - For details about the statements supported by database links, see [Table 7-168](#).
  - For details about the table types supported by database links, see [Table 7-169](#).
- **Function call using a database link**
  - When a database link calls a remote function, OUT/INOUT parameters, aggregate functions, window functions, and return set functions are not supported.
  - When a database link calls a stored procedure or function of a remote database in the PLSQL\_BODY, OUT/INOUT parameters, overloaded functions, aggregate functions, window functions, and return set functions are not supported.

---

**NOTICE**

- You can use the following syntax to call stored procedures or functions of a remote database in the PLSQL\_BODY: [CALL | SELECT] [ schema. ] { func\_name@dblink | procedure\_name@dblink } ( param\_expr )
- You can use the following syntax to call parameterless stored procedures or functions of a remote database in the PLSQL\_BODY: [CALL | SELECT] [ schema. ] { func\_name@dblink | procedure\_name@dblink } ( ).

- 
- **Synonyms**
    - The database link name cannot be created as a synonym.
    - Synonyms that point to a database link object in a remote database cannot be called through DATABASE LINK. Example:
      - Step 1: Create table **TABLE1** on **DB1**.
      - Step 2: Create **DBLINK1** on **DB2** for connecting to **DB1**. Create the synonym "CREATE SYNONYM T1 FOR TABLE1@DBLINK1".
      - Step 3: Create **DBLINK2** on **DB3** for connecting to **DB2**. Call the synonym T1 "SELECT \* FROM T1@DBLINK2" on **DB2** through **DBLINK2**.
  - **Table type constraints**

- HASHBUCKET: The query or DML operation cannot be performed on the remote hash bucket table through DATABASE LINK.
- SLICE: The query or DML operation cannot be performed on the remote slice table through DATABASE LINK.
- Replication table: The query or DML operation cannot be performed on the remote replication table through DATABASE LINK.
- TEMPORARY: The query or DML operation cannot be performed on the remote temporary table through DATABASE LINK.
- **Views**
  - Currently, you can create a view for a remote table of a database link. However, when the structure of the remote table changes, an exception may occur when you use the view. For example:
    - Step 1: Create table **TABLE1** on **DB1**.
    - Step 2: Create **DBLINK** on **DB2** for connecting to **DB1**. Create the view "CREATE VIEW V1 AS SELECT \* FROM TABLE1@DBLINK".
    - Step 3: Delete a column from **TABLE1** on **DB1**. An error is reported when you query the view on **DB2**.
- **Other scenarios**
  - The database link table does not support triggers, including the scenarios where a database link is used in the function called by a trigger, the function called by a trigger is a database link function, and the trigger is defined on a database link.
  - The UPSERT and MERGE syntaxes are not supported.
  - The CURRENT CURSOR syntax is not supported.
  - Hidden columns in a table cannot be queried.
- **Dump and backup**
  - Database objects related to database links cannot be dumped. The standby node cannot be called or connected by database links.
  - Database objects such as database links cannot be restored from the backup of their clusters. The key files of different clusters are different. Therefore, you need to use the cluster key file for decryption when using database links.
- **Predicate pushdown constraints**

The WHERE clause can only be equipped with built-in data types, operators, and functions, and the used functions are of the IMMUTABLE type.
- **Aggregate function pushdown constraints**

Only single tables that do not contain GROUP, ORDER BY, HAVING, and LIMIT clauses in the SELECT statement are supported. Window functions are not supported.
- **Hint pushdown**

Only hints in scan mode can be pushed down based on the hint conditions of database link table objects. The syntax format is as follows:

```
[no] tablescan|indexscan|indexonlyscan(table [index])
```

The table name or table alias in a query block must be unique.

**Table 7-168** Supported SQL statements

| SQL Type                  | Operation Object                                                      | Supported Option                                                                                                                                                                                                                                                                                                                                                                                                             | Execution Context                                                                        |
|---------------------------|-----------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| Creating a database link  | DATABASE LINK                                                         | N/A                                                                                                                                                                                                                                                                                                                                                                                                                          | Common transaction block                                                                 |
| Modifying a database link | DATABASE LINK                                                         | Only the username and password can be changed.                                                                                                                                                                                                                                                                                                                                                                               | Common transaction block                                                                 |
| Deleting a database link  | DATABASE LINK                                                         | N/A                                                                                                                                                                                                                                                                                                                                                                                                                          | Common transaction block                                                                 |
| SELECT statement          | Ordinary table, ordinary view, and complete-refresh materialized view | <ul style="list-style-type: none"> <li>• WHERE clause</li> <li>• JOIN clause used between a database link table and an inner table</li> <li>• JOIN clause used between database link tables</li> <li>• Aggregate function</li> <li>• LIMIT clause</li> <li>• ORDER BY clause</li> <li>• GROUP BY and HAVING clauses</li> <li>• UNION clause</li> <li>• WITH clause</li> <li>• FOR UPDATE clause</li> <li>• ROWNUM</li> </ul> | Common transaction block, stored procedure, function, advanced package, and logical view |
| INSERT statement          | Ordinary table                                                        | <ul style="list-style-type: none"> <li>• Inserting multiple values</li> </ul>                                                                                                                                                                                                                                                                                                                                                | Common transaction block, stored procedure, function, and advanced package               |
| UPDATE statement          | Ordinary table                                                        | <ul style="list-style-type: none"> <li>• LIMIT clause</li> <li>• ORDER BY clause</li> <li>• WHERE clause</li> </ul>                                                                                                                                                                                                                                                                                                          | Common transaction block, stored procedure, function, and advanced package               |

| SQL Type             | Operation Object | Supported Option                                                                                                    | Execution Context                                                          |
|----------------------|------------------|---------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| DELETE statement     | Ordinary table   | <ul style="list-style-type: none"> <li>• LIMIT clause</li> <li>• ORDER BY clause</li> <li>• WHERE clause</li> </ul> | Common transaction block, stored procedure, function, and advanced package |
| LOCK TABLE statement | Ordinary table   | <ul style="list-style-type: none"> <li>• LOCKMODE clause</li> <li>• NOWAIT clause</li> </ul>                        | Common transaction block                                                   |

**Table 7-169** Supported table types

| Dimension        | GaussDB Table Type                                                |        | Database Link Support                                               |
|------------------|-------------------------------------------------------------------|--------|---------------------------------------------------------------------|
| TEMP option      | Temporary table                                                   |        | Not supported                                                       |
|                  | Global temporary table                                            |        | Not supported                                                       |
| UNLOGGED option  | Unlogged table                                                    |        | Supported                                                           |
| Storage features | Row store                                                         | Astore | Supported                                                           |
|                  |                                                                   | Ustore | Not supported                                                       |
|                  | Partitioned table                                                 |        | Not supported                                                       |
|                  | Level-2 partitioned table                                         |        | Not supported                                                       |
| Views            | Remote view accessed by a database link                           |        | DQL statements are supported, but DML statements are not supported. |
|                  | Remote table associated with a local view through a database link |        | DQL statements are supported, but DML statements are not supported. |

## Examples

```
-- DDL statements
CREATE USER user2 WITH PASSWORD '*****'; -- Create a common user.
GRANT CREATE PUBLIC DATABASE LINK TO user2; -- Grant the permission to create database links to a user.
GRANT DROP PUBLIC DATABASE LINK TO user2; -- Grant the permission to drop database links to a user.
GRANT ALTER PUBLIC DATABASE LINK TO user2; -- Grant the permission to modify database links to a user.
REVOKE CREATE PUBLIC DATABASE LINK FROM user2; -- Revoke the permission to create database links to a user.
```

```

REVOKE DROP PUBLIC DATABASE LINK FROM user2; -- Revoke the permission to drop database links to a
user.
REVOKE ALTER PUBLIC DATABASE LINK FROM user2; -- Revoke the permission to modify database links
to a user.
CREATE PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '*****' USING (HOST
'192.168.11.11', PORT '5432', DBNAME 'db1'); -- Create a database link object
ALTER PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '*****'; -- Modify database link
information.
DROP PUBLIC DATABASE LINK; -- Drop a database link object.

-- Database link statements
-- Preparations
CREATE USER user1 WITH SYSADMIN PASSWORD '*****';
CREATE USER user2 WITH SYSADMIN PASSWORD '*****';
CREATE DATABASE db1 DBCOMPATIBILITY = 'ORA'; -- Remote database.
CREATE DATABASE db2 DBCOMPATIBILITY = 'ORA'; -- Database for testing the database link.
\c db1 user1
-- Create an ordinary table.
db1=>CREATE TABLE remote_tb(f1 int, f2 text, f3 text[]);
db1=>INSERT INTO remote_tb VALUES(0,'a',{'a0","b0","c0"});
db1=>INSERT INTO remote_tb VALUES(1,'bb',{'a1","b1","c1"});
db1=>INSERT INTO remote_tb VALUES(2,'cc',{'a2","b2","c2"});
-- Create the function CREATE OR REPLACE FUNCTION f(a in int, b in int).
db1-> return int AS tmp int := a + b;
begin
return tmp;
end;
/
CREATE FUNCTION
-- Create a synonym.
db1=>CREATE SYNONYM remote_sy FOR remote_tb;

\c db2 user2
db2=>CREATE TABLE local_tb(f1 int, f2 text, f3 text[]);
db2=>INSERT INTO local_tb VALUES (2,'c',{'a2","b2","c2"});
db2=>CREATE PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '*****' USING (HOST
'192.168.11.11', PORT '5432', DBNAME 'db1'); -- Set host and port based on actual situation.
db2=>SELECT * FROM remote_tb@dblink; -- Query the remote table.
f1 | f2 | f3
-----+-----
1 | bb | {a1,b1,c1}
2 | cc | {a2,b2,c2}
0 | a | {a0,b0,c0}
(3 rows)
db2=>INSERT INTO remote_tb@dblink VALUES (4,'d',{'a1","b2","c3"}); -- Insert data into a remote table.
INSERT 0 1
db2=>UPDATE remote_tb@dblink SET f2 = 'aa' WHERE f1 = 0; -- Update the remote table.
UPDATE 1
db2=>DELETE remote_tb@dblink DELETE f1 = 1; -- Delete data from a remote table.
DELETE 1
db2=>SELECT * FROM remote_tb@dblink JOIN local_tb ON local_tb.f1 = remote_tb.f1@dblink; -- Join a
local table to a remote table.
f1 | f2 | f3 | f1 | f2 | f3
-----+-----+-----
2 | cc | {a2,b2,c2} | 2 | c | {a2,b2,c2}
(1 row)

db2=>SELECT count(*) FROM remote_tb@dblink;
count

3
(1 row)
db2=>
db2=>SELECT f@dblink(1,2); -- Access the remote function.
f

3
(1 row)
CREATE OR REPLACE FUNCTION call_f(a in int, b in int) -- Access remote functions in PLSQL_BODY.

```

```
RETURN int AS tmp int;
begin
 tmp := f@dblink(a, b);
 return tmp;
end;
/
CREATE FUNCTION
db2=>SELECT call_f(1, 2);
call_f

 3
(1 row)
db2=>CREATE SYNONYM local_sy FOR remote_tb@dblink; -- Create a synonym for a database link object.
CREATE SYNONYM
db2=>SELECT * FROM local_sy;
f1 | f2 | f3
-----+-----
 1 | bb | {a1,b1,c1}
 2 | cc | {a2,b2,c2}
 0 | a | {a0,b0,c0}
(3 rows)
db2=>SELECT * FROM remote_sy@dblink; -- Access the remote database synonym.
f1 | f2 | f3
-----+-----
 1 | bb | {a1,b1,c1}
 2 | cc | {a2,b2,c2}
 0 | a | {a0,b0,c0}
(3 rows)
db2=>EXPLAIN VERBOSE SELECT /*+ tablescan(remote_sy) */ * FROM remote_sy@dblink; -- Partial hint
pushdown supported by the database link.
QUERY PLAN

Foreign Scan on public.remote_tb remote_sy (cost=100.00..100.03 rows=1 width=68)
 Output: f1, f2, f3
 Remote SQL: SELECT /*+ tablescan(remote_sy) */ f1, f2, f3 FROM public.remote_tb
(3 rows)

db2=>SELECT * FROM gs_database_link; -- View the database link system catalog.
db2=>START TRANSACTION;
START TRANSACTION
db2=>SELECT * FROM remote_sy@dblink;
f1 | f2 | f3
-----+-----
 1 | bb | {a1,b1,c1}
 2 | cc | {a2,b2,c2}
 0 | a | {a0,b0,c0}
(3 rows)

db2=>SELECT intransaction FROM gs_db_links; -- View the database link system view.
intransaction

t
(1 row)
db2=> END;
COMMIT
db2=>ALTER PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '*****'; -- Modify
database link information.
db2=>DROP PUBLIC DATABASE LINK dblink; -- Drop a database link object.
```

## Helpful Links

[CREATE DATABASE LINK](#), [GS\\_DB\\_LINKS](#), and [GS\\_DATABASE\\_LINK](#)

# 8 Best Practices

## 8.1 Best Practices of Table Design

### 8.1.1 Selecting a Distribution Mode

In replication mode, full data in a table is copied to each DN in the cluster. This mode is used for tables containing a small volume of data. Full data in a table stored on each DN avoids data redistribution during the join operation. This reduces network costs and plan segment (each having a thread), but generates much redundant data. Generally, this mode is only used for small dimension tables.

In hash mode, hash values are generated for one or more columns. You can obtain the storage location of a tuple based on the mapping between DNs and the hash values. In a hash table, I/O resources on each node can be used during data read/write, which improves the read/write speed of a table. Generally, a table containing a large amount data is defined as a hash table.

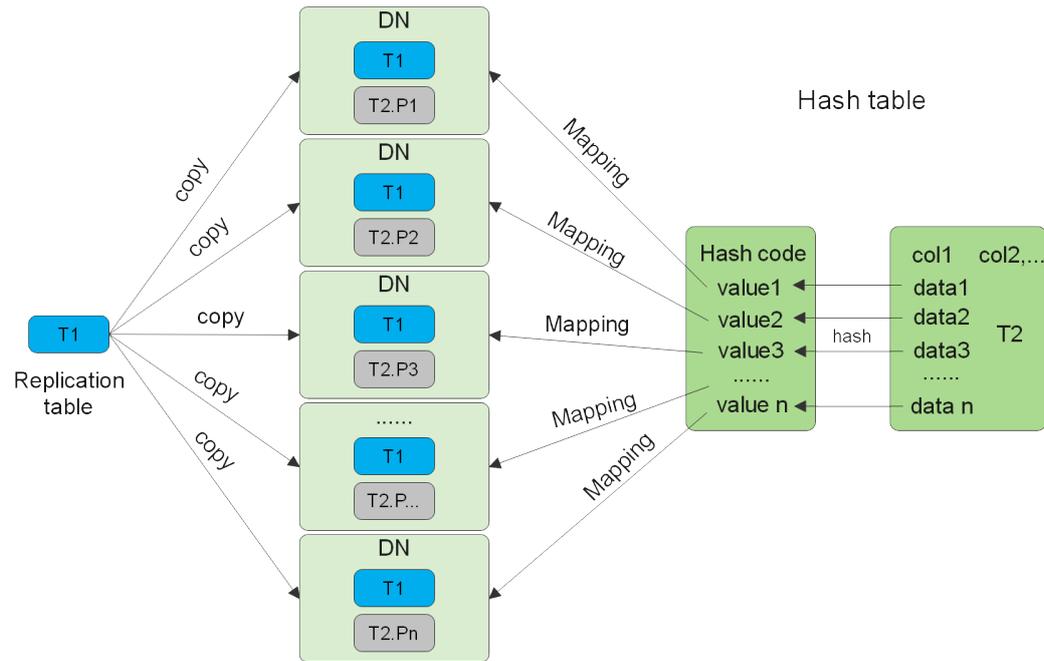
Range distribution and list distribution are user-defined distribution policies. Values in a distribution key are within a certain range or fall into a specific value range of the corresponding target DN. The two distribution modes facilitate flexible data management which, however, requires users equipped with certain data abstraction capability.

| Policy      | Description                                                                                            | Application Scenario                          |
|-------------|--------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| Hash        | Table data is distributed on all DNs in the cluster.                                                   | Fact tables containing a large amount of data |
| Replication | Full data in a table is stored on every DN in the cluster.                                             | Small tables and dimension tables             |
| Range       | Table data is mapped to specified columns based on the range and distributed to the corresponding DNs. | Users need to customize distribution rules.   |

| Policy | Description                                                                                              | Application Scenario                        |
|--------|----------------------------------------------------------------------------------------------------------|---------------------------------------------|
| List   | Table data is mapped to specified columns based on specific values and distributed to corresponding DNs. | Users need to customize distribution rules. |

As shown in **Figure 8-1**, T1 is a replication table and T2 is a hash table.

**Figure 8-1** Replication tables and hash tables



**NOTE**

- When you insert, modify, or delete data in a replication table, if you use the shippable or immutable function to encapsulate components that cannot be pushed down, data on different DNs in the replication table may be inconsistent.
- If statements with unstable results, such as window functions, rownum, and limit clauses and user-defined functions, are used to insert or modify data in a replication table, data on different nodes may be different.

## 8.1.2 Selecting Distribution Keys

Selecting a distribution key for a hash table is essential. Details are as follows:

1. **Ensure that the column values are discrete so that data can be evenly distributed to each DN.** You can select the primary key of the table as the distribution key. For example, for a person information table, choose the ID card number column as the distribution key.
2. **With the above principles met, you can select join conditions as distribution keys** so that join tasks can be pushed down to DNs, reducing the amount of data transferred between the DNs.

For a hash table, an improper distribution key may cause data skew or poor I/O performance on certain DNs. Therefore, you need to check the table to ensure that

data is evenly distributed on each DN. You can run the following SQL statements to check data skew:

```
SELECT
xc_node_id, count(1)
FROM tablename GROUP BY xc_node_id
ORDER BY xc_node_id DESC;
```

Example:

```
CREATE TABLE t1(c1 int) distribute by hash(c1);
INSERT INTO t1 values(generate_series(1,100));
select xc_node_id, count(1) from t1 group by xc_node_id order by xc_node_id desc;
DROP TABLE t1;
```

**xc\_node\_id** corresponds to a DN. Generally, **over 5% difference between the amount of data on different DNs is regarded as data skew. If the difference is over 10%, choose another distribution key.**

Multiple distribution keys can be selected in GaussDB to evenly distribute data.

You can select the distribution key of the range or list distributed table as required. In addition to selecting a proper distribution key, pay attention to the impact of distribution rules on data distribution.

## 8.1.3 Using Partitioned Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored on these physical partitions, instead of the logical partitioned table. A partitioned table has the following advantages over an ordinary table:

1. High query performance: You can specify partitions when querying partitioned tables, improving query efficiency.
2. High availability: If a certain partition in a partitioned table is faulty, data in the other partitions is still available.
3. Easy maintenance: To fix a partitioned table having a faulty partition, you only need to fix the partition.

GaussDB supports range partitioned tables, list partitioned tables, and hash partitioned tables.

- Range partitioned table: Data in different ranges is mapped to different partitions. The range is determined by the partition key specified during the partitioned table creation. The partition key is usually a date. For example, sales data is partitioned by month.
- List partitioned table: Key values contained in the data are stored in different partitions, and the data is mapped to each partition in sequence. The key values contained in the partitions are specified when the partitioned table is created.
- Hash partitioned table: Data is mapped to each partition based on the internal hash algorithm. The number of partitions is specified when the partitioned table is created.

## 8.1.4 Selecting a Data Type

Use the following principles to select efficient data types:

1. **Select data types that facilitate data calculation.**  
Generally, the calculation of integers (including common comparison calculations, such as =, >, <, ≥, ≤, and ≠, and GROUP BY) is more efficient than that of strings and floating point numbers.
2. **Select data types with a short length.**  
Data types with short length reduce both the data file size and the memory used for computing, improving the I/O and computing performance. For example, use **SMALLINT** instead of **INT**, and **INT** instead of **BIGINT**.
3. **Use the same data type for a join.**  
You are advised to use the same data type for a join. To join columns with different data types, the database needs to convert them to the same type, which leads to additional performance overheads.

## 8.1.5 Checking a Node Where a Table Resides

When creating a table, you can specify how the table is distributed or replicated among nodes. For details, see [•DISTRIBUTEBY](#). For details about distribution modes, see [Selecting a Distribution Mode](#).

When creating a table, you can also set **Node Group** to specify a group to which the table belongs. For details, see [•TO{GROUPgroupname}|...](#)

You can also run the following command to view the instance where the table is located.

1. Query the schema to which the table belongs.  

```
select t1.nspname,t2.relname from pg_namespace t1,pg_class t2 where t1.oid = t2.relnamespace and t2.relname = 'table1';
```

In the preceding command, **nspname** indicates the name of a schema, **relname** indicates the name of a table, an index, or a view, **oid** indicates the row identifier, **relnamespace** is the OID of the namespace that contains the relationship, and **table1** indicates a table name.

2. Check **relname** and **nodeoids** of the table.  

```
select t1.relname,t2.nodeoids from pg_class t1, pgxc_class t2, pg_namespace t3 where t1.relfilenode = t2.pcrelid and t1.relnamespace=t3.oid and t1.relname = 'table1' and t3.nspname = 'schema1';
```

In the preceding command, **nodeoids** indicates the OID list of the nodes where the table is distributed, **relfilenode** indicates the name of the file related to the table on the disk, **pcrelid** indicates the OID of the table, and **schema1** indicates the schema of the table queried in step 1.

3. Query the instance where the table is located based on the queried node where the table is distributed.

```
select * from pgxc_node where oid in (nodeoids1, nodeoids2, nodeoids3);
```

In the preceding command, **nodeoids1**, **nodeoids2**, **nodeoids3** indicates the three nodeoids queried in step 2. Use the actual nodeoids and separate them with commas (,).

## 8.2 Best Practices of SQL Queries

Based on the SQL execution mechanism and a large number of practices, SQL statements can be optimized by following certain rules to enable the database to execute SQL statements more quickly and obtain correct results.

- Replace UNION with UNION ALL.  
UNION eliminates duplicate rows while merging two result sets but UNION ALL merges the two result sets without deduplication. Therefore, replace UNION with UNION ALL if you are sure that the two result sets do not contain duplicate rows based on the service logic.
- Add NOT NULL to the JOIN columns.  
If there are many **NULL** values in the JOIN columns, you can add the filter criterion IS NOT NULL to filter data in advance to improve the JOIN efficiency.
- Replace NOT IN with NOT EXISTS.  
Nested loop anti join must be used to implement NOT IN, and hash anti join is required for NOT EXISTS. If no NULL value exists in the JOIN columns, NOT IN is equivalent to NOT EXISTS. Therefore, if you are sure that no NULL value exists, you can convert NOT IN to NOT EXISTS to generate hash join and to improve the query performance.

The statements for creating a foreign table are as follows:

```
DROP SCHEMA IF EXISTS no_in_to_no_exists_test CASCADE;
CREATE SCHEMA no_in_to_no_exists_test;
SET CURRENT_SCHEMA=no_in_to_no_exists_test;
CREATE TABLE t1(c1 int, c2 int, c3 int);
CREATE TABLE t2(d1 int, d2 int NOT NULL, d3 int);
```

The statement for implementing the query using NOT IN is as follows:

```
SELECT * FROM t1 WHERE c1 NOT IN (SELECT d2 FROM t2);
```

The plan is as follows:

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 NOT IN (SELECT d2 FROM t2);
QUERY PLAN

Streaming (type: GATHER) (cost=0.06..38.57 rows=3 width=12)
Node/s: All datanodes
-> Nested Loop Anti Join (cost=0.00..38.44 rows=3 width=12)
Join Filter: ((t1.c1 = t2.d2) OR (t1.c1 IS NULL))
-> Seq Scan on t1 (cost=0.00..14.14 rows=30 width=12)
-> Materialize (cost=0.00..18.08 rows=90 width=4)
-> Streaming(type: BROADCAST) (cost=0.00..17.93 rows=90 width=4)
Spawn on: All datanodes
-> Seq Scan on t2 (cost=0.00..14.14 rows=30 width=4)
(9 rows)
```

Because there is no null value in the **t2.d2** column (the **t2.d2** column is **NOT NULL** in the table definition), the query can be equivalently modified as follows:

```
SELECT * FROM t1 WHERE NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
```

The generated plan is as follows:

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
QUERY PLAN

Streaming (type: GATHER) (cost=14.38..29.99 rows=3 width=12)
Node/s: All datanodes
-> Hash Right Anti Join (cost=14.32..29.86 rows=3 width=12)
Hash Cond: (t2.d2 = t1.c1)
-> Streaming(type: REDISTRIBUTE) (cost=0.00..15.49 rows=30 width=4)
Spawn on: All datanodes
-> Seq Scan on t2 (cost=0.00..14.14 rows=30 width=4)
-> Hash (cost=14.14..14.14 rows=29 width=12)
-> Seq Scan on t1 (cost=0.00..14.14 rows=30 width=12)
(9 rows)
```

- Use hashagg.

If the GROUP BY condition exists in the query statement, the generated plan may contain sorting operations, that is, the plan contains the GroupAgg+Sort operator. As a result, the performance is poor. You can set the GUC parameter **work\_mem** to increase the available memory and generate a plan with HashAgg to avoid sorting operations and improve performance. For details about how to set **work\_mem**, contact the administrator.

- Replace functions with CASE statements.  
The GaussDB performance greatly deteriorates if a large number of functions are called. In this case, you can modify the pushdown functions to CASE statements.
- Do not use functions or expressions for indexes.  
Using functions or expressions for indexes stops indexing. Instead, it enables scanning on the full table.
- Do not use != or <> operators, **NULL**, **OR**, or implicit parameter conversion in **WHERE** clauses.
- Prevent SQL statements from triggering FATAL errors.  
If service statements triggers FATAL errors, threads are rebuilt. Rectify them promptly to prevent the issue from escalating to an anomaly. If too many of such statements exist, a large number of threads exit, causing the following problems:
  - The database SQL statement execution and connection performance are affected.
  - Threads are frequently created and destroyed. As a result, the thread pool usage increases sharply.
  - Service connections are interrupted. If services repeatedly retry, the system pressure increases rapidly, affecting the overall performance.
- Split complex SQL statements.  
You can split an SQL statement into several ones and save the execution result to a temporary table if the SQL statement is too complex to be tuned using the solutions above, including but not limited to the following scenarios:
  - The same subquery is involved in multiple SQL statements of a job and the subquery contains large amounts of data.
  - Incorrect plan cost causes a small hash bucket of subquery. For example, the actual number of rows is 10 million, but only 1000 rows are in hash bucket.
  - Functions such as substr and to\_number cause incorrect measures for subqueries containing a large amount of data.
  - **BROADCAST** subqueries are performed on large tables in multi-DN environment.

For details about optimization, see [Typical SQL Optimization Methods](#).

## 8.3 Best Practices for Data Skew Query

## 8.3.1 Quickly Locating Tables That Cause Data Skew

Currently, the [table\\_distribution\(schemaname text, tablename text\)](#) and [table\\_distribution\(\)](#) functions as well as the [PGXC\\_GET\\_TABLE\\_SKEWNESS](#) view are provided to query for data skew. You can choose any of them as needed.

### Scenario 1: Data Skew Caused by a Full Disk

First, use the [pg\\_stat\\_get\\_last\\_data\\_changed\\_time\(oid\)](#) function to query for the tables whose data is changed recently. The last change time of a table is recorded only on the CN where INSERT, UPDATE, and DELETE operations are performed. Therefore, you need to query for tables that are changed within the last day (the period can be changed in the function).

```
CREATE OR REPLACE FUNCTION get_last_changed_table(OUT schemaname text, OUT relname text)
RETURNS setof record
AS $$
DECLARE
 row_data record;
 row_name record;
 query_str text;
 query_str_nodes text;
BEGIN
 query_str_nodes := 'SELECT node_name FROM pgxc_node where node_type = "C"';
 FOR row_name IN EXECUTE(query_str_nodes) LOOP
 query_str := 'EXECUTE DIRECT ON (' || row_name.node_name || ') "SELECT b.nspname,a.relname
FROM pg_class a INNER JOIN pg_namespace b on a.relnamespace = b.oid where
pg_stat_get_last_data_changed_time(a.oid) BETWEEN current_timestamp - 1 AND current_timestamp;"';
 FOR row_data IN EXECUTE(query_str) LOOP
 schemaname = row_data.nspname;
 relname = row_data.relname;
 return next;
 END LOOP;
 END LOOP;
 return;
END; $$
LANGUAGE 'plpgsql';
```

Then, execute the [table\\_distribution\(schemaname text, tablename text\)](#) function to query for the storage space occupied the tables on each DN.

```
SELECT table_distribution(schemaname,relname) FROM get_last_changed_table();
```

### Scenario 2: Routine Data Skew Inspection

- If the number of tables in the database is less than 10,000, use the skew view to query data skew of all tables in the database.
- If the number of tables in the database is no less than 10,000, you are advised to use the [table\\_distribution\(\)](#) function instead of the [PGXC\\_GET\\_TABLE\\_SKEWNESS](#) view because the view takes a longer time (hours) due to the query of the entire database for skew columns. When you use the [table\\_distribution\(\)](#) function, you can define the output based on [PGXC\\_GET\\_TABLE\\_SKEWNESS](#), optimizing the calculation and reducing the output columns. For example:

```
SELECT schemaname,tablename,max(dnsize) AS maxsize, min(dnsize) AS minsize
FROM pg_catalog.pg_class c
INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
INNER JOIN pg_catalog.table_distribution() s ON s.schemaname = n.nspname AND s.tablename =
c.relname
```

```
INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND x.plocator = 'H'
GROUP BY schemaname,tablename;
```

# 9 User-defined Functions

---

When the cluster is started, CNs and DN and the UDF master process are also started. To execute a UDF in fenced mode, the UDF master process forks itself to UDF worker processes, and UDF worker processes execute the UDF in fenced mode.

## 9.1 PL/SQL Functions

PL/SQL is a loadable procedural language.

Functions created using PL/SQL can be used in any place where you can use built-in functions. For example, you can create calculation functions with complex conditions and use them to define operators or use them for index expressions.

SQL is used by most databases as a query language. It is portable and easy to learn. Each SQL statement must be executed independently by a database server.

The client application performs the following processes for each query: send a query to the database server, wait for the query to be received, receive and process the result, perform related calculation, and then send more queries to the server. If the client and the database server are not on the same machine, this process also causes inter-process communication and network load.

PL/SQL enables a whole computing part and a series of queries to be grouped inside a database server. This makes procedural language available and SQL easier to use. In addition, the client/server communication cost is reduced.

- Extra round-trip communication between clients and servers is eliminated.
- Intermediate results that are not required by clients do not need to be sorted or transmitted between the clients and servers.
- Parsing can be skipped in multiple rounds of queries.

PL/SQL can use all data types, operators, and functions in SQL. For details about the PL/SQL syntax for creating functions, see [CREATE FUNCTION](#).

PL/SQL is a loadable procedural language. Its application method is similar to that of [Stored Procedures](#). The difference is that [Stored Procedures](#) has no return value, and PL/SQL functions have return values.

XML data can be used as the input parameter, output parameter, user-defined variable, and return value of a user-defined function.

# 10 Stored Procedures

---

## 10.1 Overview

In GaussDB, business rules and logics are saved as stored procedures.

A stored procedure is a combination of SQL, PL/SQL, and Java statements. Stored procedures can move the code that executes business rules from applications to databases. Therefore, the code storage can be used by multiple programs at a time.

For details about how to create and call a stored procedure, see [CREATE PROCEDURE](#).

The application methods for PL/SQL functions mentioned in [PL/SQL Functions](#) are the same as those for stored procedures. Unless otherwise specified, the following sections apply to stored procedures and PL/SQL functions.

## 10.2 Data Types

A data type refers to a value set and an operation set defined on the value set. GaussDB consists of tables, each of which is defined by its own columns. Each column corresponds to a data type. GaussDB uses corresponding functions to perform operations on data based on data types. For example, GaussDB can perform operations such as addition, subtraction, multiplication, and division on numeric data.

XML data can be used as input parameters, output parameters, user-defined variables, and return values of stored procedures, as well as stored procedures that support autonomous transactions.

## 10.3 Data Type Conversion

Certain data types in the database support implicit data type conversions, such as assignments and parameters called by functions. For other data types (such as int), you can use the type conversion functions provided by GaussDB, such as the CAST function, to forcibly convert them.

**Table 10-1** lists common implicit data type conversions in GaussDB.

**NOTICE**

The valid value range of **DATE** supported by GaussDB is from 4713 B.C. to 294276 A.D.

**Table 10-1** Implicit data type conversions

| Raw Data Type | Target Data Type | Remarks                                      |
|---------------|------------------|----------------------------------------------|
| CHAR          | VARCHAR2         | -                                            |
| CHAR          | NUMBER           | Raw data must consist of digits.             |
| CHAR          | DATE             | Raw data cannot exceed the valid date range. |
| CHAR          | RAW              | -                                            |
| CHAR          | CLOB             | -                                            |
| VARCHAR2      | CHAR             | -                                            |
| VARCHAR2      | NUMBER           | Raw data must consist of digits.             |
| VARCHAR2      | DATE             | Raw data cannot exceed the valid date range. |
| VARCHAR2      | CLOB             | -                                            |
| NUMBER        | CHAR             | -                                            |
| NUMBER        | VARCHAR2         | -                                            |
| DATE          | CHAR             | -                                            |
| DATE          | VARCHAR2         | -                                            |
| RAW           | CHAR             | -                                            |
| RAW           | VARCHAR2         | -                                            |
| CLOB          | CHAR             | -                                            |
| CLOB          | VARCHAR2         | -                                            |
| CLOB          | NUMBER           | Raw data must consist of digits.             |
| INT4          | CHAR             | -                                            |

## 10.4 DECLARE Syntax

### 10.4.1 Basic Structure

#### Structure

A PL/SQL block can contain a sub-block which can be placed in any section. The following describes the architecture of a PL/SQL block:

- Declaration section: declares variables, types, cursors, and regional stored procedures and functions used in the PL/SQL block.

DECLARE

#### NOTE

This section is optional if no variables need to be declared.

- An anonymous block may omit the **DECLARE** keyword if no variable needs to be declared.
- For a stored procedure, **AS** is used, which is equivalent to **DECLARE**. The **AS** keyword must be reserved even if there is no variable declaration section.
- Execution section: specifies procedure and SQL statements. It is the main section of a program and is mandatory.  
BEGIN
- Exception-handling section: processes errors. It is optional.  
EXCEPTION
- End. Mandatory.  
END;  
/

#### NOTICE

You are not allowed to use consecutive tabs in the PL/SQL block because they may result in an exception when the **gsq** tool is executed with the **-r** parameter specified.

#### Types

PL/SQL blocks are classified into the following types:

- Anonymous block: a dynamic block that can be executed only for once. For details about the syntax, see [Figure 10-1](#).
- Subprogram: a stored procedure, function, operator, or advanced package stored in a database. A subprogram created in a database can be called by other programs.

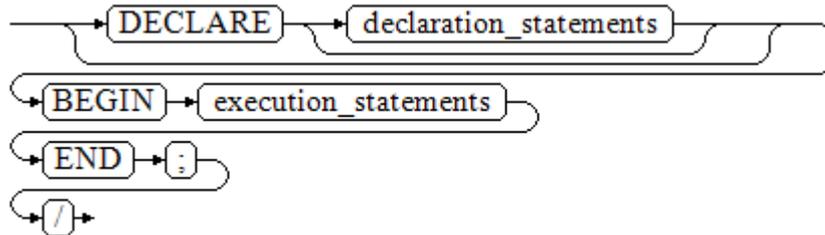
### 10.4.2 Anonymous Blocks

An anonymous block applies to a script infrequently executed or a one-off activity. An anonymous block is executed in a session and is not stored.

## Syntax

Figure 10-1 shows the syntax diagrams for an anonymous block.

Figure 10-1 anonymous\_block::=



Details about the syntax diagram are as follows:

- The execution section of an anonymous block starts with a **BEGIN** statement, has a break with an **END** statement, and ends with a semicolon (;). Type a slash (/) and press **Enter** to execute the statement.

### NOTICE

The terminator "/" must be written in an independent row.

- The declaration section includes the variable definition, type, and cursor definition.
- A simplest anonymous block does not execute any commands. However, at least one statement, even a **NULL** statement, must be presented in any implementation blocks.

## Examples

The following lists basic anonymous block programs:

```
-- Null statement block
gaussdb=# BEGIN
 NULL;
END;
/

-- Display information on the console.
gaussdb=# BEGIN
 db_output.print_line('hello world!');
END;
/
hello world!
ANONYMOUS BLOCK EXECUTE

-- Display variable content on the console.
gaussdb=# DECLARE
 my_var VARCHAR2(30);
BEGIN
 my_var := 'world';
 db_output.print_line('hello'||my_var);
END;
/
helloworld
ANONYMOUS BLOCK EXECUTE
```

## 10.4.3 Subprograms

A subprogram stores stored procedures, functions, operators, and advanced packages. A subprogram created in a database can be called by other programs.

## 10.5 Basic Statements

During PL/SQL programming, you may define some variables, assign values to variables, and call other stored procedures. The following sections describe basic PL/SQL statements, including variable definition statements, value assignment statements, call statements, and return statements.

### NOTE

You are not advised to call the SQL statements containing passwords in the stored procedures because authorized users may view the stored procedure file in the database and password information is leaked. If a stored procedure contains other sensitive information, permission to access this procedure must be configured, preventing information leakage.

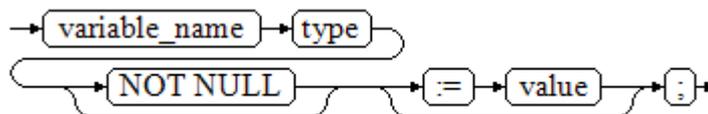
### 10.5.1 Variable Definition Statements

This section describes the declaration of variables in the PL/SQL and the scope of this variable in codes.

#### Variable Declaration

For details about the variable declaration syntax, see [Figure 10-2](#).

Figure 10-2 declare\_variable::=



The above syntax diagram is explained as follows:

- *variable\_name*: specifies the name of a variable.
- **type** indicates the type of a variable.
- **value** indicates the initial value of the variable. (If the initial value is not given, NULL is taken as the initial value.) **value** can also be an expression.

#### Example

```
gaussdb=# DECLARE
emp_id INTEGER := 7788; -- Define a variable and assign a value to it.
BEGIN
emp_id := 5*7784; -- Assign a value to the variable.
END;
/
```

In addition to the declaration of basic variable types, **%TYPE** and **%ROWTYPE** can be used to declare variables related to table columns or table structures.

### %TYPE Attribute

**%TYPE** declares a variable to be of the same data type as a previously declared variable (for example, a column in a table). For example, if you want to define a *my\_name* variable whose data type is the same as the data type of the **firstname** column in the **employee** table, you can define the variable as follows:

```
my_name employee.firstname%TYPE
-- Example
DROP TABLE IF EXISTS employee;
NOTICE: table "employee" does not exist, skipping
DROP TABLE
CREATE TABLE employee(firstname varchar,secondname varchar);
CREATE TABLE
DECLARE
 my_name employee.firstname%TYPE;
BEGIN
 my_name = 'abc';
 DBE_OUTPUT.PRINT_LINE(my_name);
END;
/
abc
ANONYMOUS BLOCK EXECUTE
```

In this way, you can declare *my\_name* without the need of knowing the data type of **firstname** in **employee**, and the data type of *my\_name* can be automatically updated when the data type of **firstname** changes.

```
TYPE employee_record is record (id INTEGER, firstname VARCHAR2(20));
my_employee employee_record;
my_id my_employee.id%TYPE;
my_id_copy my_id%TYPE;
-- Example
DECLARE
 TYPE employee_record is record (id INTEGER, firstname VARCHAR2(20));
 my_employee employee_record;
 my_id my_employee.id%TYPE;
 my_id_copy my_id%TYPE;
BEGIN
 my_employee.id := 1;
 my_employee.firstname := 'ab2';
 my_id = 2;
 my_id_copy = 3;
 DBE_OUTPUT.PRINT_LINE(my_employee.id);
 DBE_OUTPUT.PRINT_LINE(my_employee.firstname);
 DBE_OUTPUT.PRINT_LINE(my_id);
 DBE_OUTPUT.PRINT_LINE(my_id_copy);
END;
/
1
ab2
2
3
ANONYMOUS BLOCK EXECUTE
```

### %ROWTYPE Attribute

**%ROWTYPE** declares data types of a set of data. It stores a row of table data or results fetched from a cursor. For example, if you want to define a set of data with the same column names and column data types as the **employee** table, you can define the data as follows:

```
my_employee employee%ROWTYPE
-- Example
DROP TABLE IF EXISTS employee;
DROP TABLE
CREATE TABLE employee(firstname varchar,secondname varchar);
```

```
DECLARE
 my_employee employee%ROWTYPE;
BEGIN
 my_employee.firstname := 'ab1';
 my_employee.seconname := 'ab2';
 DBE_OUTPUT.PRINT_LINE(my_employee.firstname);
 DBE_OUTPUT.PRINT_LINE(my_employee.seconname);
END;
/
ab1
ab2
ANONYMOUS BLOCK EXECUTE
```

#### NOTE

- In the environment with multiple CNs, the **%ROWTYPE** and **%TYPE** attributes of the temporary table cannot be declared in a stored procedure. The temporary table is valid only in the current session. During compilation, other CNs cannot view the temporary table of the current CN. Therefore, if there are multiple CNs, the system displays a message indicating that the temporary table does not exist.
- If **%TYPE** is used to obtain the type from *record.column* and such type is defined by **%TYPE**, the constraint on the original type (such as NUMBER(3) and VARCHAR2(10)) is not retained.

## Scope of a Variable

The scope of a variable indicates the accessibility and availability of the variable in code block. In other words, a variable takes effect only within its scope.

- A variable must be declared in the declaration section of a **BEGIN-END** block. The necessity of such declaration is also determined by block structure, which requires that a variable has different scopes and lifetime during a process.
- A variable can be defined multiple times in different scopes, and inner definition can cover outer one.
- A variable defined in an outer block can also be used in a nested block. However, the outer block cannot access variables in the nested block.

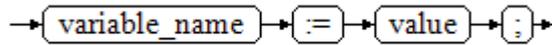
### Example

```
gaussdb=# DECLARE
 emp_id INTEGER :=7788; -- Define a variable and assign a value to it.
 outer_var INTEGER :=6688; -- Define a variable and assign a value to it.
BEGIN
 DECLARE
 emp_id INTEGER :=7799; -- Define a variable and assign a value to it.
 inner_var INTEGER :=6688; -- Define a variable and assign a value to it.
 BEGIN
 db_output.print_line('inner emp_id =||emp_id); -- Display the value 7799.
 db_output.print_line('outer_var =||outer_var); -- Reference a variable of an outer block.
 END;
 db_output.print_line('outer emp_id =||emp_id); -- Display the value 7788.
END;
/
```

## 10.5.2 Assignment Statements

### Variable Syntax

**Figure 10-3** shows the syntax diagram for assigning a value to a variable.

**Figure 10-3** assignment\_value::=

The above syntax diagram is explained as follows:

- *variable\_name*: specifies the name of a variable.
- *value* can be a value or an expression. The type of *value* must be compatible with the type of *variable\_name*.

## Variable Value Assignment Example

```
gaussdb=# DECLARE
emp_id INTEGER := 7788; -- Assignment
BEGIN
emp_id := 5; -- Assignment
DBE_OUTPUT.PRINT_LINE(emp_id);
emp_id := 5*7784;
DBE_OUTPUT.PRINT_LINE(emp_id);
END;
/
-- The result is as follows:
5
38920
ANONYMOUS BLOCK EXECUTE
```

## INTO/BULK COLLECT INTO

**INTO** and **BULK COLLECT INTO** store values returned by statements in a stored procedure to variables. **BULK COLLECT INTO** allows some or all returned values to be temporarily stored in an array.

### Example

```
gaussdb=# DROP TABLE IF EXISTS customers;
NOTICE: table "customers" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE customers(id int,name varchar);
CREATE TABLE
gaussdb=# INSERT INTO customers VALUES(1,'ab');
gaussdb=# DECLARE
my_id integer;
BEGIN
select id into my_id from customers limit 1; -- Assign a value.
END;
/
ANONYMOUS BLOCK EXECUTE
gaussdb=# DECLARE
type id_list is varray(6) of customers.id%type;
id_arr id_list;
BEGIN
select id bulk collect into id_arr from customers order by id DESC limit 20; -- Assign values in batches.
END;
/
ANONYMOUS BLOCK EXECUTE
```

**NOTICE**

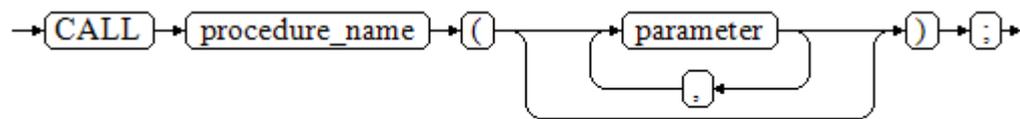
- BULK COLLECT INTO can only assign values to arrays or collections in batches. The array type uses LIMIT properly to prevent performance deterioration caused by excessive operations on data.
- For array variables, elements in parentheses () are preferentially identified as index sets. Therefore, expressions with parentheses cannot be written after array variables. For example, **select (1+3) into va(5)** cannot be written as **select into va(5) (1+3)** or **select into va[5] (1+3)**.
- BULK COLLECT INTO can be used only in the ORA-compatible database.

## 10.5.3 Call Statements

### Syntax

Figure 10-4 shows the syntax diagram for calling a clause.

Figure 10-4 call\_clause::=



The above syntax diagram is explained as follows:

- *procedure\_name*: specifies the name of a stored procedure.
- *parameter*: specifies the parameters for the stored procedure. You can set no parameter or multiple parameters.

### Examples

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
gaussdb=# CREATE SCHEMA hr;
gaussdb=# SET CURRENT_SCHEMA = hr;
gaussdb=# CREATE TABLE staffs
(
 section_id NUMBER,
 first_name VARCHAR2,
 phone_number VARCHAR2,
 salary NUMBER
);
gaussdb=# INSERT INTO staffs VALUES (30, 'mike', '13567829252', 5800);
gaussdb=# INSERT INTO staffs VALUES (40, 'john', '17896354637', 4000);

-- Create the stored procedure proc_staffs.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_staffs
(
 section NUMBER(6),
 salary_sum out NUMBER(8,2),
 staffs_count out INTEGER
)
IS
BEGIN
 SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM hr.staffs where section_id = section;
END;
```

```
/
CREATE PROCEDURE
-- Create the stored procedure proc_return.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_return
AS
v_num NUMBER(8,2);
v_sum INTEGER;
BEGIN
proc_staffs(30, v_sum, v_num); --Call a statement.
dbe_output.print_line(v_sum||'#'||v_num);
RETURN; -- Return a statement.
END;
/

-- Call the stored procedure proc_return.
gaussdb=# CALL proc_return();
5800#1.00
proc_return

(1 row)

-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE proc_staffs;
gaussdb=# DROP PROCEDURE proc_return;

-- Create the function func_return.
gaussdb=# CREATE OR REPLACE FUNCTION func_return returns void
language plpgsql
AS $$
DECLARE
v_num INTEGER := 1;
BEGIN
dbe_output.print_line(v_num);
RETURN; -- Return a statement.
END $$;
CREATE FUNCTION

-- Call the function func_return.
gaussdb=# CALL func_return();
1
func_return

(1 row)

-- Drop the function.
gaussdb=# DROP FUNCTION func_return;
```

## 10.6 Dynamic Statements

### 10.6.1 Executing Dynamic Query Statements

You can perform dynamic queries using **EXECUTE IMMEDIATE** or **OPEN FOR** in GaussDB. **EXECUTE IMMEDIATE** dynamically executes **SELECT** statements and **OPEN FOR** combines use of cursors. If you need to store query results in a dataset, use **OPEN FOR**.

#### EXECUTE IMMEDIATE

[Figure 10-5](#) shows the syntax diagram.

Figure 10-5 EXECUTE IMMEDIATE dynamic\_select\_clause::=

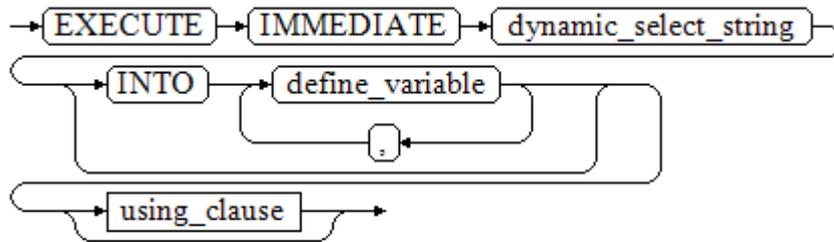
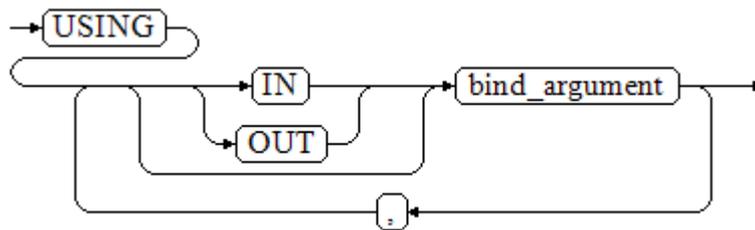


Figure 10-6 shows the syntax diagram for using\_clause.

Figure 10-6 using\_clause::=



The above syntax diagram is explained as follows:

- *define\_variable*: specifies a variable to store single-row query results.
- **USING IN** *bind\_argument*: specifies the variable whose value is passed to the dynamic SQL statement. The variable is used when a dynamic placeholder exists in *dynamic\_select\_string*.
- **USING OUT** *bind\_argument*: specifies the variable that stores a value returned by the dynamic SQL statement.

**NOTICE**

- In query statements, **INTO** and **OUT** cannot coexist.
- A placeholder name starts with a colon (:) and can be followed by digits, characters, or character strings (quoted digits, characters, or character strings cannot be used). A placeholder name corresponds to *bind\_argument* in the USING clause.
- *bind\_argument* can only be a value, variable, or expression. It cannot be a database object such as a table name, column name, and data type. That is, *bind\_argument* cannot be used to transfer schema objects for dynamic SQL statements. If a stored procedure needs to transfer database objects through *bind\_argument* to construct dynamic SQL statements (generally, DDL statements), you are advised to use double vertical bars (||) to concatenate *dynamic\_select\_clause* with a database object.
- A dynamic PL/SQL block allows duplicate placeholders. That is, a placeholder can correspond to only one *bind\_argument* in the USING clause. When the GUC parameter **behavior\_compat\_options** is set to **dynamic\_sql\_compat**, the bind arguments in the USING clause are matched in sequence based on the placeholder sequence. Duplicate placeholders will not be identified as the same placeholder.

**Example:**

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
gaussdb=# CREATE SCHEMA hr;
gaussdb=# SET CURRENT_SCHEMA = hr;
gaussdb=# CREATE TABLE staffs
(
 staff_id NUMBER,
 first_name VARCHAR2,
 salary NUMBER
);
gaussdb=# INSERT INTO staffs VALUES (200, 'mike', 5800);
gaussdb=# INSERT INTO staffs VALUES (201, 'lily', 3000);
gaussdb=# INSERT INTO staffs VALUES (202, 'john', 4400);

-- Retrieve values from dynamic statements (INTO clause).
gaussdb=# DECLARE
 staff_count VARCHAR2(20);
BEGIN
 EXECUTE IMMEDIATE 'select count(*) from hr.staffs'
 INTO staff_count;
 db_output.print_line(staff_count);
END;
/
3
ANONYMOUS BLOCK EXECUTE
-- Pass and retrieve values (the INTO clause is used before the USING clause).
gaussdb=# CREATE OR REPLACE PROCEDURE dynamic_proc
AS
 staff_id NUMBER(6) := 200;
 first_name VARCHAR2(20);
 salary NUMBER(8,2);
BEGIN
 EXECUTE IMMEDIATE 'select first_name, salary from hr.staffs where staff_id = :1'
 INTO first_name, salary
 USING IN staff_id;
 db_output.print_line(first_name || ' ' || salary);
END;
/
CREATE PROCEDURE
```

```
-- Call the stored procedure.
gaussdb=# CALL dynamic_proc();
mike 5800.00
dynamic_proc

(1 row)

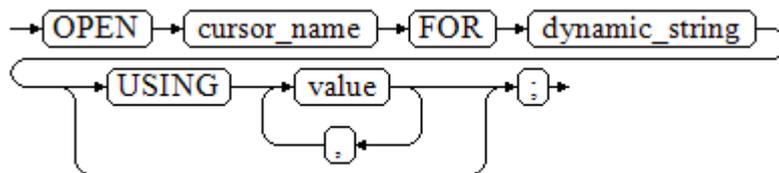
-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE dynamic_proc;
gaussdb=# DROP TABLE staffs;
```

## OPEN FOR

Dynamic query statements can be executed by using **OPEN FOR** to open dynamic cursors.

**Figure 10-7** shows the syntax diagram.

**Figure 10-7** open\_for::=



Parameter description:

- *cursor\_name*: specifies the name of the cursor to be opened.
- *dynamic\_string*: specifies the dynamic query statement.
- **USING value**: applies when a placeholder exists in *dynamic\_string*.

For details about use of cursors, see [Cursors](#).

Example:

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
gaussdb=# CREATE SCHEMA hr;
gaussdb=# SET CURRENT_SCHEMA = hr;
gaussdb=# CREATE TABLE staffs
(
 section_id NUMBER,
 first_name VARCHAR2,
 phone_number VARCHAR2,
 salary NUMBER
);
gaussdb=# INSERT INTO staffs VALUES (30, 'mike', '13567829252', 5800);
gaussdb=# INSERT INTO staffs VALUES (40, 'john', '17896354637', 4000);

gaussdb=# DECLARE
 name VARCHAR2(20);
 phone_number VARCHAR2(20);
 salary NUMBER(8,2);
 sqlstr VARCHAR2(1024);

 TYPE app_ref_cur_type IS REF CURSOR; -- Define the cursor type.
 my_cur app_ref_cur_type; -- Define the cursor variable.

BEGIN
 sqlstr := 'select first_name,phone_number,salary from hr.staffs
```

```

where section_id = :1';
OPEN my_cur FOR sqlstr USING '30'; -- Open the cursor. USING is optional.
FETCH my_cur INTO name, phone_number, salary; -- Retrieve the data.
WHILE my_cur%FOUND LOOP
 db_output.print_line(name||'#'||phone_number||'#'||salary);
 FETCH my_cur INTO name, phone_number, salary;
END LOOP;
CLOSE my_cur; -- Close the cursor.
END;
/
mike#13567829252#5800.00
ANONYMOUS BLOCK EXECUTE

```

## 10.6.2 Executing Dynamic Non-Query Statements

### Syntax

Figure 10-8 shows the syntax diagram.

Figure 10-8 noselect::=

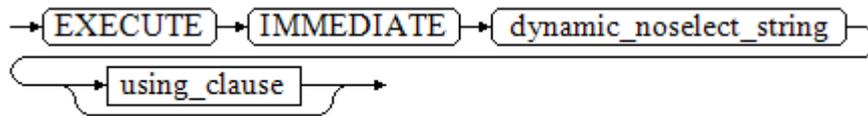
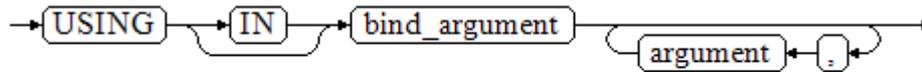


Figure 10-9 shows the syntax diagram for using\_clause.

Figure 10-9 using\_clause::=



The above syntax diagram is explained as follows:

**USING IN** *bind\_argument* is used to specify the variable whose value is passed to the dynamic SQL statement. The variable is used when a placeholder exists in *dynamic\_noselect\_string*. That is, a placeholder is replaced by the corresponding *bind\_argument* when a dynamic SQL statement is executed. Note that *bind\_argument* can only be a value, variable, or expression, and cannot be a database object such as a table name, column name, and data type. If a stored procedure needs to transfer database objects through *bind\_argument* to construct dynamic SQL statements (generally, DDL statements), you are advised to use double vertical bars (||) to concatenate *dynamic\_select\_clause* with a database object. In addition, a dynamic PL/SQL block allows duplicate placeholders. That is, a placeholder can correspond to only one *bind\_argument*. When the GUC parameter **behavior\_compat\_options** is set to **dynamic\_sql\_compat**, *bind\_argument* in the USING clause is matched in the placeholder sequence. Duplicate placeholders will not be identified as the same placeholder (placeholder names cannot contain quoted digits, characters, or strings).

### Examples

```

-- Create a table.
gaussdb=# CREATE TABLE sections_t1

```

```
(
 section NUMBER(4) ,
 section_name VARCHAR2(30),
 manager_id NUMBER(6),
 place_id NUMBER(4)
)
DISTRIBUTE BY hash(manager_id);

-- Declare a variable.
gaussdb=# DECLARE
 section NUMBER(4) := 280;
 section_name VARCHAR2(30) := 'Info support';
 manager_id NUMBER(6) := 103;
 place_id NUMBER(4) := 1400;
 new_colname VARCHAR2(10) := 'sec_name';
BEGIN
-- Execute the query.
 EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :4)'
 USING section, section_name, manager_id, place_id;
-- Execute the query (duplicate placeholders).
 EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :1)'
 USING section, section_name, manager_id;
-- Run the ALTER statement. You are advised to use double vertical bars (||) to concatenate the dynamic
DDL statement with a database object.
 EXECUTE IMMEDIATE 'alter table sections_t1 rename section_name to ' || new_colname;
END;
/

-- Query data.
gaussdb=# SELECT * FROM sections_t1;
 section | sec_name | manager_id | place_id
-----+-----+-----+-----
 280 | Info support | 103 | 1400
 280 | Info support | 103 | 280
(2 rows)

-- Delete the table.
gaussdb=# DROP TABLE sections_t1;
```

### 10.6.3 Dynamically Calling Stored Procedures

This section describes how to dynamically call store procedures. You must use anonymous statement blocks to package stored procedures or statement blocks and append **IN** and **OUT** behind the **EXECUTE IMMEDIATE...USING** statement to input and output parameters.

#### Syntax

Figure 10-10 shows the syntax diagram.

Figure 10-10 call\_procedure::=

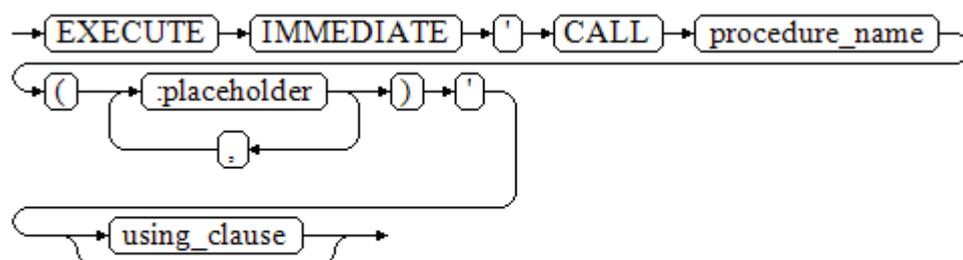
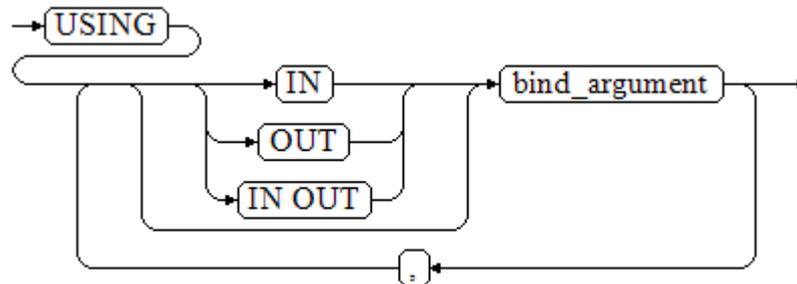


Figure 10-11 shows the syntax diagram for `using_clause`.

Figure 10-11 `using_clause::=`



The above syntax diagram is explained as follows:

- **CALL *procedure\_name***: calls the stored procedure.
- **[:*placeholder1*;*placeholder2*, ...]**: indicates the list of placeholders for stored procedure parameters. Placeholder names cannot contain quoted digits, characters, or character strings. The numbers of the placeholders and parameters are the same.
- **USING [IN|OUT|IN OUT] *bind\_argument***: specifies the variable whose value is passed to the stored procedure parameter. The modifiers in front of **bind\_argument** and of the corresponding parameter are the same.
- Overloaded functions or stored procedures with placeholders cannot be called.

## Examples

```
-- Create the stored procedure proc_add.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_add
(
 param1 in INTEGER,
 param2 out INTEGER,
 param3 in INTEGER
)
AS
BEGIN
 param2:= param1 + param3;
END;
/

gaussdb=# DECLARE
input1 INTEGER:=1;
input2 INTEGER:=2;
statement VARCHAR2(200);
param2 INTEGER;
BEGIN
 -- Declare the call statement.
 statement := 'call proc_add(:col_1, :col_2, :col_3)';
 -- Execute the statement.
 EXECUTE IMMEDIATE statement
 USING IN input1, OUT param2, IN input2;
 db_output.print_line('result is: '||to_char(param2));
END;
/
result is: 3
ANONYMOUS BLOCK EXECUTE
-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE proc_add;
```

## 10.6.4 Dynamically Calling Anonymous Blocks

This section describes how to execute anonymous blocks in dynamic statements. Append **IN** and **OUT** behind the **EXECUTE IMMEDIATE...USING** statement to input and output parameters.

### Syntax

Figure 10-12 shows the syntax diagram.

Figure 10-12 call\_anonymous\_block::=

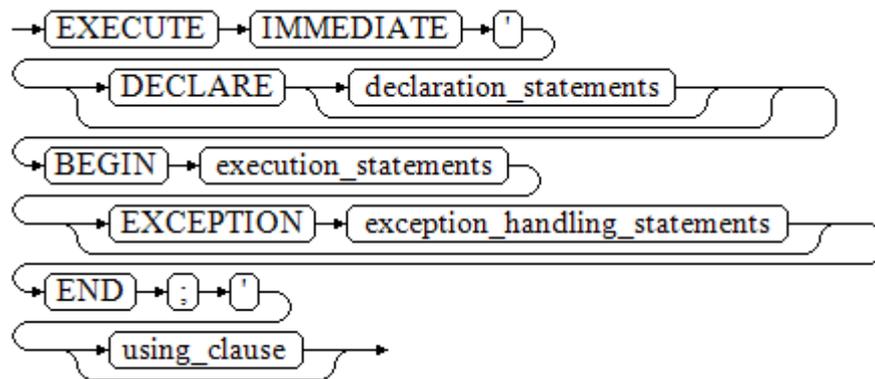
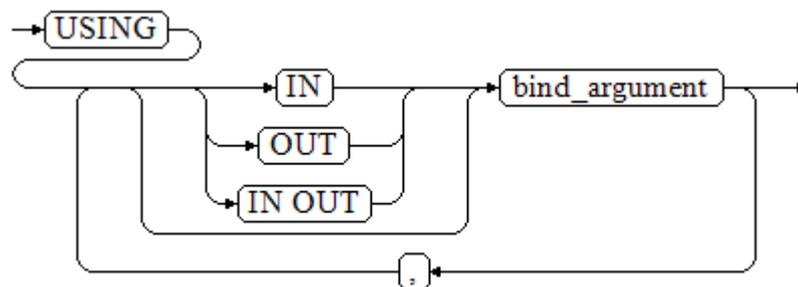


Figure 10-13 shows the syntax diagram for **using\_clause**.

Figure 10-13 using\_clause::=



The above syntax diagram is explained as follows:

- The execute part of an anonymous block starts with a **BEGIN** statement, has a break with an **END** statement, and ends with a semicolon (;).
- **USING [IN|OUT|IN OUT]bind\_argument**: specifies where the variable passed to the stored procedure parameter value is stored. The modifiers in front of **bind\_argument** and of the corresponding parameter are the same.
- The input and output parameters in the middle of an anonymous block are designated by placeholders. The number of placeholders must match the number of parameters. Placeholder names cannot contain quoted digits, characters, or strings. The sequences of the parameters corresponding to the placeholders and the **USING** parameters are the same.

- Currently in GaussDB, when dynamic statements call anonymous blocks, placeholders cannot be used to pass input and output parameters in an **EXCEPTION** statement.
- Overloaded functions with placeholders cannot be called.
- The PERFORM keyword cannot be used to call a stored procedure when parameters are bound.
- Variables declared in an anonymous block and binding parameters cannot be used in the same statement at the same time.
- Output parameters cannot be bound when the SELECT INTO statement in an anonymous block calls FUNCTION/PROCEDURE that contains output parameters.
- Only SQL statements called in anonymous blocks and stored procedures with OUT/INOUT parameters can be bound to parameters. For example, expressions and cursors are used in anonymous blocks, and dynamic statements are called in nested mode in anonymous blocks.
- When the **dynamic\_sql\_check** parameter is enabled and the number of placeholders is the same as the number of parameters, an error is reported if placeholders with the same name are used as anonymous block parameters. In this case, you need to rename the placeholders.

## Examples

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
gaussdb=# CREATE SCHEMA hr;
gaussdb=# SET CURRENT_SCHEMA = hr;
gaussdb=# CREATE TABLE staffs
(
 staff_id NUMBER,
 first_name VARCHAR2,
 salary NUMBER
);
gaussdb=# INSERT INTO staffs VALUES (200, 'mike', 5800);
gaussdb=# INSERT INTO staffs VALUES (201, 'lily', 3000);
gaussdb=# INSERT INTO staffs VALUES (202, 'john', 4400);

--Create the stored procedure dynamic_proc.
gaussdb=# CREATE OR REPLACE PROCEDURE dynamic_proc
AS
 staff_id NUMBER(6) := 200;
 first_name VARCHAR2(20);
 salary NUMBER(8,2);
BEGIN
 --Execute the anonymous block.
 EXECUTE IMMEDIATE 'begin select first_name, salary into :first_name, :salary from hr.staffs where
staff_id= :dno; end;'
 USING OUT first_name, OUT salary, IN staff_id;
 db_output.print_line(first_name|| ' ' || salary);
END;
/

-- Call the stored procedure.
gaussdb=# CALL dynamic_proc();
mike 5800.00
dynamic_proc

(1 row)
-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE dynamic_proc;

-- Example of an error reported when dynamic_sql_check is enabled
```

```
gaussdb=# SET behavior_compat_options = 'dynamic_sql_check';
SET
gaussdb=# CREATE OR REPLACE PROCEDURE test_proc_exception001(a out integer, b inout integer, c
integer)
as
BEGIN
a := 1;
begin b := 1/0; end;
EXCEPTION
WHEN others THEN
b := 2;
END;
/
CREATE PROCEDURE
gaussdb=#
DECLARE
a integer := 1;
c integer;
BEGIN
execute immediate 'begin test_proc_exception001(:1,:2,:1); end;' using in out a, out c, a;
END;
/
ERROR: argnum not match in Dynamic SQL, using args num : 3 , actual sql args num : 2
CONTEXT: PL/pgSQL function inline_code_block line 4 at EXECUTE statement
-- Change the placeholders with the same name.
gaussdb=#
DECLARE
a integer := 1;
c integer;
BEGIN
execute immediate 'begin test_proc_exception001(:1,:2,:3); end;' using in out a, out c, a;
END;
/
ANONYMOUS BLOCK EXECUTE
gaussdb=# DROP PROCEDURE test_proc_exception001;
DROP PROCEDURE
gaussdb=# reset behavior_compat_options;
```

## 10.7 Control Statements

### 10.7.1 RETURN Statements

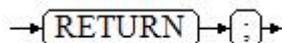
In GaussDB, data can be returned in either of the following ways: **RETURN**, **RETURN NEXT**, or **RETURN QUERY**. **RETURN NEXT** and **RETURN QUERY** are used only for functions and cannot be used for stored procedures.

#### 10.7.1.1 RETURN

##### Syntax

Figure 10-14 shows the syntax diagram for a return statement.

Figure 10-14 return\_clause::=



The above syntax diagram is explained as follows:

This statement returns control from a stored procedure or function to a caller.

## Examples

See [Examples](#) for call statement examples.

### 10.7.1.2 RETURN NEXT and RETURN QUERY

#### Syntax

When creating a function, specify **SETOF** *datatype* for the return values.

return\_next\_clause::=



return\_query\_clause::=



The above syntax diagram is explained as follows:

If a function needs to return a result set, use **RETURN NEXT** or **RETURN QUERY** to add results to the result set, and then continue to execute the next statement of the function. As the **RETURN NEXT** or **RETURN QUERY** statement is executed repeatedly, more and more results will be added to the result set. After the function is executed, all results are returned.

**RETURN NEXT** can be used for scalar and compound data types.

**RETURN QUERY** has a variant **RETURN QUERY EXECUTE**. You can add dynamic queries and add parameters to the queries by **USING**.

## Examples

```
gaussdb=# DROP TABLE t1;
gaussdb=# CREATE TABLE t1(a int);
gaussdb=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
gaussdb=# CREATE OR REPLACE FUNCTION fun_for_return_next() RETURNS SETOF t1 AS $$
DECLARE
 r t1%ROWTYPE;
BEGIN
 FOR r IN select * from t1
 LOOP
 RETURN NEXT r;
 END LOOP;
 RETURN;
END;
$$ LANGUAGE plpgsql;
gaussdb=# call fun_for_return_next();
 a

 1
10
(2 rows)

-- RETURN QUERY
gaussdb=# CREATE OR REPLACE FUNCTION fun_for_return_query() RETURNS SETOF t1 AS $$
DECLARE
```

```

r t1%ROWTYPE;
BEGIN
 RETURN QUERY select * from t1;
END;
$$
language plpgsql;
gaussdb=# call fun_for_return_query();
 a

 1
 10
(2 rows)

```

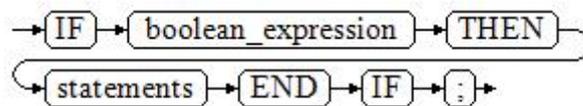
## 10.7.2 Conditional Statements

Conditional statements are used to decide whether given conditions are met. Operations are executed based on the decisions made.

GaussDB supports five usages of **IF**:

- **IF\_THEN**

**Figure 10-15** IF\_THEN::=



**IF\_THEN** is the simplest form of **IF**. If the condition is true, statements are executed. If it is false, they are skipped.

Example:

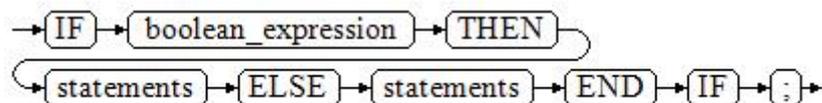
```

gaussdb=#
DECLARE
 v_user_id integer default 1;
BEGIN
 IF v_user_id <> 0 THEN
 raise info 'v_user_id is NOT 0';
 END IF;
END;
/
INFO: v_user_id is NOT 0
ANONYMOUS BLOCK EXECUTE

```

- **IF\_THEN\_ELSE**

**Figure 10-16** IF\_THEN\_ELSE::=



**IF-THEN-ELSE** statements add **ELSE** branches and can be executed if the condition is false.

Example:

```

gaussdb=#
DECLARE
 v_user_id integer default 0;

```

```

BEGIN
 IF v_user_id <> 0 THEN
 raise info 'v_user_id is NOT 0';
 ELSE
 raise info 'v_user_id is 0';
 END IF;
END;
/
INFO: v_user_id is 0
ANONYMOUS BLOCK EXECUTE

```

- IF\_THEN\_ELSE IF

IF statements can be nested in the following way:

```

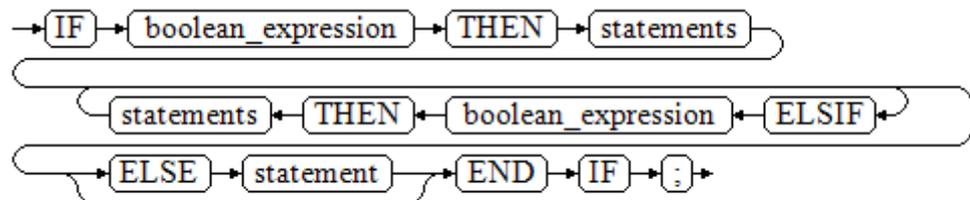
gaussdb=#
DECLARE
 v_user_id integer default 1;
BEGIN
 IF v_user_id = 0 THEN
 raise info 'v_user_id is 0';
 ELSE
 IF v_user_id > 0 THEN
 raise info 'v_user_id > 0';
 END IF;
 END IF;
END;
/
INFO: v_user_id > 0
ANONYMOUS BLOCK EXECUTE

```

Actually, this is a way of an IF statement nesting in the ELSE part of another IF statement. Therefore, an END IF statement is required for each nesting IF statement and another END IF statement is required to end the parent IF-ELSE statement. To set multiple options, use the following form:

- IF\_THEN\_ELSIF\_ELSE

Figure 10-17 IF\_THEN\_ELSIF\_ELSE::=



Example:

```

gaussdb=# DECLARE
 v_user_id integer default NULL;
BEGIN
 IF v_user_id = 0 THEN
 raise info 'v_user_id is 0';
 ELSIF v_user_id > 0 THEN
 raise info 'v_user_id > 0';
 ELSIF v_user_id < 0 THEN
 raise info 'v_user_id < 0';
 ELSE
 raise info 'v_user_id is NULL';
 END IF;
END;
/
INFO: v_user_id is NULL
ANONYMOUS BLOCK EXECUTE

```

- IF\_THEN\_ELSEIF\_ELSE

**ELSEIF** is an alias of **ELSIF**.

Example:

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_control_structure(i in integer)
AS
BEGIN
 IF i > 0 THEN
 raise info 'i:% is greater than 0. ',i;
 ELSIF i < 0 THEN
 raise info 'i:% is smaller than 0. ',i;
 ELSE
 raise info 'i:% is equal to 0. ',i;
 END IF;
 RETURN;
END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_control_structure(3);
INFO: i:3 is greater than 0.
proc_control_structure

(1 row)

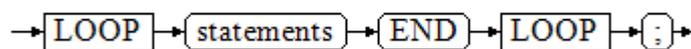
-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE proc_control_structure;
DROP PROCEDURE
```

## 10.7.3 Loop Statements

### Simple LOOP Statements

Syntax diagram

Figure 10-18 loop::=



Example

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
BEGIN
 count:=0;
 LOOP
 IF count > i THEN
 raise info 'count is %. ', count;
 EXIT;
 ELSE
 count:=count+1;
 END IF;
 END LOOP;
END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_loop(10,5);
```

```
INFO: count is 11.
count

 11
(1 row)
```

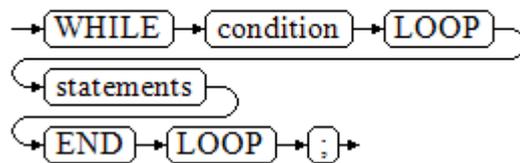
**NOTICE**

The loop must be exploited together with **EXIT**; otherwise, a dead loop occurs.

## WHILE\_LOOP Statements

### Syntax diagram

Figure 10-19 while\_loop::=



If the conditional expression is true, a series of statements in the WHILE statement are repeatedly executed and the condition is decided each time the loop body is executed.

### Example

```
gaussdb=# CREATE TABLE integertable(c1 integer) DISTRIBUTE BY hash(c1);
CREATE TABLE
gaussdb=# CREATE OR REPLACE PROCEDURE proc_while_loop(maxval in integer)
AS
 DECLARE
 i int :=1;
 BEGIN
 WHILE i < maxval LOOP
 INSERT INTO integertable VALUES(i);
 i:=i+1;
 END LOOP;
 END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL proc_while_loop(10);
proc_while_loop

(1 row)

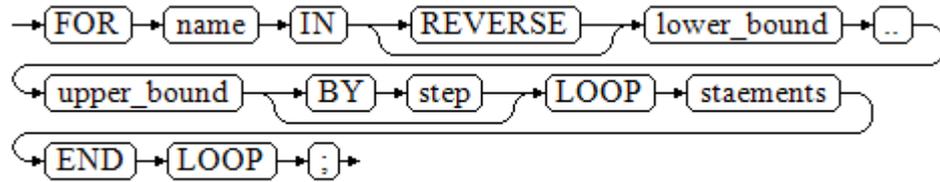
-- Drop the stored procedure and table.
gaussdb=# DROP PROCEDURE proc_while_loop;
DROP PROCEDURE

gaussdb=# DROP TABLE integertable;
DROP TABLE
```

## FOR\_LOOP (*Integer variable*) Statement

### Syntax diagram

Figure 10-20 for\_loop::=



### NOTE

- The variable **name** is automatically defined as the **integer** type and exists only in this loop. The variable name falls between lower\_bound and upper\_bound.
- When the keyword **REVERSE** is used, the lower bound must be greater than or equal to the upper bound; otherwise, the loop body is not executed.

### Example

```
-- Loop from 0 to 5.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_for_loop()
AS
 BEGIN
 FOR I IN 0..5 LOOP
 DBE_OUTPUT.PRINT_LINE('It is '||to_char(I) || ' time;');
 END LOOP;
 END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL proc_for_loop();
It is 0 time;
It is 1 time;
It is 2 time;
It is 3 time;
It is 4 time;
It is 5 time;
proc_for_loop

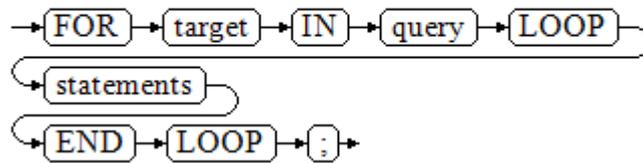
(1 row)

-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE proc_for_loop;
DROP PROCEDURE
```

## FOR\_LOOP Query Statements

### Syntax diagram

Figure 10-21 for\_loop\_query::=



**NOTE**

- The variable **target** is automatically defined, its type is the same as that in the **query** result, and it is valid only in this loop. The **target** value is the query result.
- You can use EXECUTE to add a dynamic query and use USING to insert parameters to the query. For details, see the dynamic query content in [Example](#).

**Example**

```

-- Display the query result from the loop.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_for_loop_query()
AS
 record VARCHAR2(50);
BEGIN
 FOR record IN SELECT spcname FROM pg_tablespace LOOP
 db_output.print_line(record);
 END LOOP;
END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL proc_for_loop_query();
pg_default
pg_global
proc_for_loop_query

(1 row)

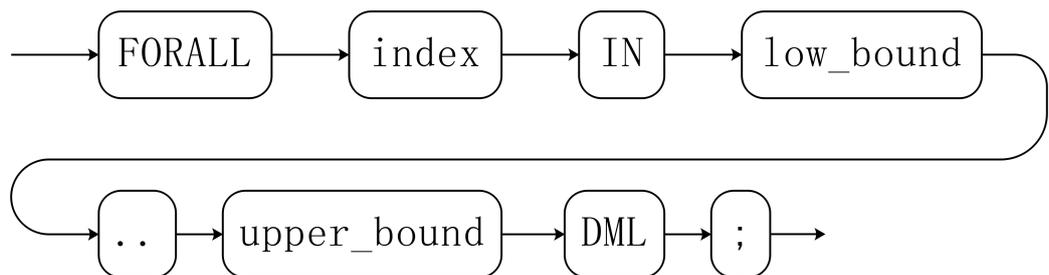
-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE proc_for_loop_query;
DROP PROCEDURE

```

**FORALL Batch Query Statements**

**Syntax diagram**

Figure 10-22 forall::=



 NOTE

The variable **index** is automatically defined as the **integer** type and exists only in this loop. The index value falls between `low_bound` and `upper_bound`.

**Example**

```
gaussdb=# CREATE TABLE TEST_t1 (
 title NUMBER(6),
 did VARCHAR2(20),
 data_period VARCHAR2(25),
 kind VARCHAR2(25),
 interval VARCHAR2(20),
 time DATE,
 isModified VARCHAR2(10)
)
DISTRIBUTE BY hash(did);
CREATE TABLE

gaussdb=# INSERT INTO TEST_t1 VALUES(8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833',
to_date('21-06-1999', 'dd-mm-yyyy'), 'SH_CLERK');
INSERT 0 1
gaussdb=# CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
 FORALL i IN 100..120
 update TEST_t1 set title = title + 100*i;
END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL proc_forall();
proc_forall

(1 row)

-- Query the calling result of the stored procedure.
gaussdb=# SELECT * FROM TEST_t1;
title | did | data_period | kind | interval | time | ismodified
-----+-----+-----+-----+-----+-----+-----
231008 | Donald | OConnell | DOCONNEL | 650.507.9833 | 1999-06-21 00:00:00 | SH_CLERK
(1 row)

-- Drop the stored procedure and table.
gaussdb=# DROP PROCEDURE proc_forall;
DROP PROCEDURE

gaussdb=# DROP TABLE TEST_t1;
DROP TABLE
```

## 10.7.4 Branch Statements

### Syntax

[Figure 10-23](#) shows the syntax diagram for a return statement.

Figure 10-23 case\_when::=

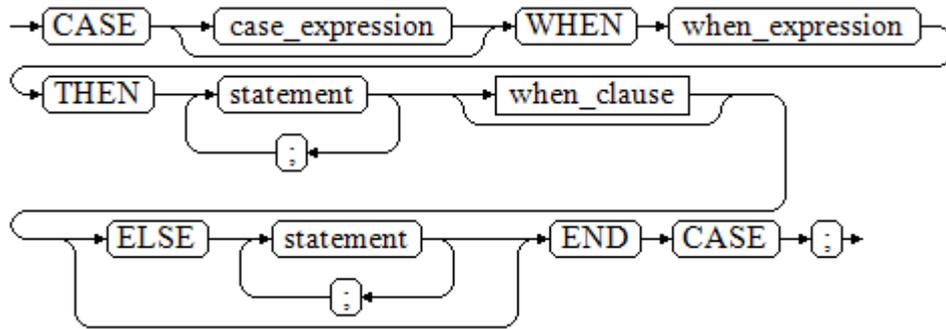
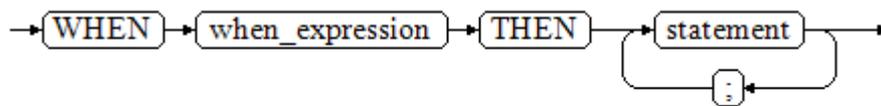


Figure 10-24 shows the syntax diagram for when\_clause.

Figure 10-24 when\_clause::=



Parameters:

- *case\_expression*: specifies the variable or expression.
- *when\_expression*: specifies the constant or conditional expression.
- *statement*: specifies the statement to be executed.

## Examples

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_case_branch(pi_result in integer, pi_return out integer)
AS
BEGIN
CASE pi_result
WHEN 1 THEN
pi_return := 111;
WHEN 2 THEN
pi_return := 222;
WHEN 3 THEN
pi_return := 333;
WHEN 6 THEN
pi_return := 444;
WHEN 7 THEN
pi_return := 555;
WHEN 8 THEN
pi_return := 666;
WHEN 9 THEN
pi_return := 777;
WHEN 10 THEN
pi_return := 888;
ELSE
pi_return := 999;
END CASE;
raise info 'pi_return : %',pi_return ;
END;
/
CREATE PROCEDURE
gaussdb=# CALL proc_case_branch(3,0);
```

```
INFO: pi_return : 333
pi_return

 333
(1 row)

-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE proc_case_branch;
DROP PROCEDURE
```

## 10.7.5 NULL Statements

In PL/SQL programs, **NULL** statements are used to indicate "nothing should be done", equal to placeholders. They grant meanings to some statements and improve program readability.

### Syntax

The following shows example use of **NULL** statements.

```
DECLARE
...
BEGIN
...
 IF v_num IS NULL THEN
 NULL; -- No data needs to be processed.
 END IF;
END;
/
```

### Examples

```
gaussdb=# DECLARE
 v_num integer default NULL;
BEGIN
 IF v_num IS NOT NULL THEN
 raise info 'v_num is NULL';
 ELSE
 NULL; -- No data needs to be processed.
 END IF;
END;
/
ANONYMOUS BLOCK EXECUTE
```

## 10.7.6 Error Trapping Statements

By default, any error occurring in a PL/SQL function aborts execution of the function, and indeed of the surrounding transaction as well. You can trap errors and restore from them by using a **BEGIN** block with an **EXCEPTION** clause. The syntax is an extension of the normal syntax for a **BEGIN** block:

```
[<<label>>]
[DECLARE
 declarations]
BEGIN
 statements
EXCEPTION
 WHEN condition [OR condition ...] THEN
 handler_statements
 [WHEN condition [OR condition ...] THEN
 handler_statements
 ...]
END;
```

If no error occurs, this form of block simply executes all the statements, and then control passes to the next statement after **END**. But if an error occurs within the

statements, further processing of the statements is abandoned, and control passes to the **EXCEPTION** list. The list is searched for the first condition matching the error that occurred. If a match is found, the corresponding **handler\_statements** are executed, and then control passes to the next statement after **END**. If no match is found, the error propagates out as though the **EXCEPTION** clause were not there at all: Error codes can be used to catch other error codes of the same type.

The error can be caught by an enclosing block with **EXCEPTION**, or if there is none it aborts processing of the function.

The condition names can be any of those shown in *Error Code Reference*. The special condition name **OTHERS** matches every error type except **QUERY\_CANCELED**.

If a new error occurs within the selected **handler\_statements**, it cannot be caught by this **EXCEPTION** clause, but is propagated out. A surrounding **EXCEPTION** clause could catch it.

When an error is caught by an **EXCEPTION** clause, the local variables of the PL/SQL function remain as they were when the error occurred, but all changes to persistent database state within the block are rolled back.

Example:

```
gaussdb=# CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20)) DISTRIBUTE BY
hash(id);
CREATE TABLE

gaussdb=# INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
INSERT 0 1

gaussdb=# CREATE FUNCTION fun_exp() RETURNS INT
AS $$
DECLARE
 x INT :=0;
 y INT;
BEGIN
 UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
 x := x + 1;
 y := x / 0;
EXCEPTION
 WHEN division_by_zero THEN
 RAISE NOTICE 'caught division_by_zero';
 RETURN x;
END;$$
LANGUAGE plpgsql;
CREATE FUNCTION

gaussdb=# call fun_exp();
NOTICE: caught division_by_zero
 fun_exp

 1
(1 row)

gaussdb=# select * from mytab;
 id | firstname | lastname
-----+-----+-----
 1 | Tom | Jones
(1 row)

gaussdb=# DROP FUNCTION fun_exp();
DROP FUNCTION
```

```
gaussdb=# DROP TABLE mytab;
DROP TABLE
```

When control reaches the assignment to **y**, it will fail with a **division\_by\_zero** error. This will be caught by the **EXCEPTION** clause. The value returned in the **RETURN** statement will be the incremented value of **x**.

#### NOTE

A block containing an **EXCEPTION** clause is more expensive to enter and exit than a block without one. Therefore, do not use **EXCEPTION** without need.

In the following scenario, an exception cannot be caught, and the entire transaction rolls back. The threads of the nodes participating the stored procedure exit abnormally due to node failure and network fault, or the source data is inconsistent with that of the table structure of the target table during the COPY FROM operation.

#### Example: Exceptions with **UPDATE/INSERT**

This example uses exception handling to perform either **UPDATE** or **INSERT**, as appropriate:

```
gaussdb=# CREATE TABLE db (a INT, b TEXT);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
 LOOP

 -- First try to update the key
 UPDATE db SET b = data WHERE a = key;
 IF found THEN
 RETURN;
 END IF;

 -- The key does not exist. Therefore, try to insert one. If someone else inserts the same key concurrently, the
 unique key may fail to be obtained.
 BEGIN
 INSERT INTO db(a,b) VALUES (key, data);
 RETURN;
 EXCEPTION WHEN unique_violation THEN
 -- Do nothing, and loop to try the UPDATE again.
 END;
 END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE FUNCTION

gaussdb=# SELECT merge_db(1, 'david');
merge_db

(1 row)

gaussdb=# SELECT merge_db(1, 'dennis');
merge_db

(1 row)

--Drop the function and table.
gaussdb=# DROP FUNCTION merge_db;
DROP FUNCTION

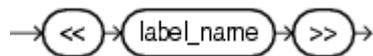
gaussdb=# DROP TABLE db;
DROP TABLE
```

## 10.7.7 GOTO Statements

A GOTO statement unconditionally transfers the control from the current statement to a labeled statement. The GOTO statement changes the execution logic. Therefore, use this statement only when necessary. Alternatively, you can use the EXCEPTION statement to handle issues in special scenarios. To run a GOTO statement, the labeled statement must be unique.

### Syntax

label declaration ::=



goto statement ::=



### Examples

```
gaussdb=# CREATE OR REPLACE PROCEDURE GOTO_test()
AS
DECLARE
 v1 int;
BEGIN
 v1 := 0;
 LOOP
 EXIT WHEN v1 > 100;
 v1 := v1 + 2;
 if v1 > 25 THEN
 GOTO pos1;
 END IF;
 END LOOP;
<<pos1>>
v1 := v1 + 10;
raise info 'v1 is %.', v1;
END;
/
CREATE PROCEDURE
gaussdb=# call GOTO_test();
INFO: v1 is 36.
goto_test

(1 row)
```

### Constraints

Using GOTO statements has the following constraints:

- A GOTO statement does not allow multiple labeled statements even if the statements are in different blocks.

```
BEGIN
 GOTO pos1;
<<pos1>>
SELECT * FROM ...
<<pos1>>
UPDATE t1 SET ...
END;
```

- A GOTO statement cannot transfer control to the IF, CASE, or LOOP statement.

```
BEGIN
 GOTO pos1;
 IF valid THEN
 <<pos1>>
 SELECT * FROM ...
 END IF;
END;
```

- A GOTO statement cannot transfer control from one IF clause to another, or from one WHEN clause in the CASE statement to another.

```
BEGIN
 IF valid THEN
 GOTO pos1;
 SELECT * FROM ...
 ELSE
 <<pos1>>
 UPDATE t1 SET ...
 END IF;
END;
```

- Transferring control from an outer block to an inner BEGIN-END block is not supported.

```
BEGIN
 GOTO pos1;
 BEGIN
 <<pos1>>
 UPDATE t1 SET ...
 END;
END;
```

- Transferring control from an exception handler to the current BEGIN-END block is not supported. However, transferring control to the upper-layer BEGIN-END block is supported.

```
BEGIN
 <<pos1>>
 UPDATE t1 SET ...
 EXCEPTION
 WHEN condition THEN
 GOTO pos1;
END;
```

- To branch to a position that does not have an executable statement, add the NULL statement.

```
DECLARE
 done BOOLEAN;
BEGIN
 FOR i IN 1..50 LOOP
 IF done THEN
 GOTO end_loop;
 END IF;
 <<end_loop>> -- not allowed unless an executable statement follows
 NULL; -- add NULL statement to avoid error
 END LOOP; -- raises an error without the previous NULL
END;
/
```

## 10.8 Transaction Statements

A stored procedure itself is automatically in a transaction. A transaction is automatically started when the most peripheral stored procedure is called. In addition, the transaction is automatically committed when the calling ends, or is rolled back when an exception occurs during calling. In addition to automatic transaction control, you can also use COMMIT/ROLLBACK to control transactions in stored procedures. Running the COMMIT/ROLLBACK commands in a stored procedure will commit or roll back the current transaction and automatically starts

a new transaction. All subsequent operations will be performed in the new transaction.

A savepoint is a special mark inside a transaction. It allows all commands that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint. In a stored procedure, you can use savepoints to manage transactions. Currently, you can create, roll back, and release savepoints. If a savepoint for rollback is used in a stored procedure, only the modification of the current transaction is rolled back. The execution process of the stored procedure is not changed, and the values of local variables in the stored procedure are not rolled back.

---

#### NOTICE

The applicable contexts are as follows:

1. COMMIT, ROLLBACK, and SAVEPOINT can be used in PL/SQL stored procedures/functions.
2. COMMIT, ROLLBACK, and SAVEPOINT can be used in stored procedures and functions that contain EXCEPTION.
3. COMMIT, ROLLBACK, and SAVEPOINT can be used in EXCEPTION statements of stored procedures.
4. A stored procedure that contains COMMIT, ROLLBACK, or SAVEPOINT (which means the stored procedure is controlled by BEGIN, START, or END) can be called in a transaction block.
5. A stored procedure that contains savepoints can be called in a subtransaction and an externally defined savepoint is used to roll back the transaction to the savepoint defined outside the stored procedure.
6. A savepoint defined in the stored procedure can be viewed outside the stored procedure. That is, the modification of the transaction can be rolled back to the savepoint defined in the stored procedure.
7. COMMIT, ROLLBACK, and SAVEPOINT, as well as IF, FOR, CURSOR LOOP, and WHILE, can be called in most PL/SQL contexts and statements.

The following content can be committed or rolled back:

1. DDL statements after COMMIT or ROLLBACK can be committed or rolled back.
  2. DML statements after COMMIT or ROLLBACK can be committed.
  3. GUC parameters in stored procedures can be committed or rolled back.
- 

## Limitations

The unapplicable contexts are as follows:

1. COMMIT, ROLLBACK, and SAVEPOINT cannot be called in stored procedures other than PL/SQL, such as PL/Java and PL/Python.
2. After SAVEPOINT is called in a transaction block, stored procedures that contain COMMIT/ROLLBACK cannot be called.
3. Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in TRIGGER.

4. COMMIT, ROLLBACK, and SAVEPOINT cannot be called in EXECUTE statements.
5. Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in CURSOR statements.
6. Stored procedures that contain IMMUTABLE or SHIPPABLE cannot call COMMIT, ROLLBACK, SAVEPOINT or another stored procedure that contain COMMIT, ROLLBACK, or SAVEPOINT.
7. Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in SQL statements other than SELECT PROC and CALL PROC.
8. COMMIT, ROLLBACK, or SAVEPOINT cannot be called in a stored procedure whose header contains GUC parameters.
9. COMMIT, ROLLBACK, or SAVEPOINT cannot be called in expressions or CURSOR and EXECUTE statements.
10. Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in the return values and expression calculation of stored procedures.
11. Savepoints defined outside a stored procedure cannot be released in the stored procedure.
12. A stored procedure transaction and its autonomous transaction are two independent transactions and cannot use savepoints defined in each other's transaction.
13. Advanced packages cannot call stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT statements through DBE\_SQL.

The following content cannot be committed or rolled back:

1. Variables declared or imported in stored procedures cannot be committed or rolled back.
2. In stored procedures, GUC parameters that take effect only after a restart cannot be submitted or rolled back.

## Syntax

```
Define a savepoint.
SAVEPOINT savepoint_name;
Roll back a savepoint.
ROLLBACK TO [SAVEPOINT] savepoint_name;
Release a savepoint.
RELEASE [SAVEPOINT] savepoint_name;
```

## Examples

### NOTE

COMMIT/ROLLBACK can be used in PL/SQL stored procedures.

```
gaussdb=# CREATE TABLE EXAMPLE1(COL1 INT);

gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE()
AS
BEGIN
FOR i IN 0..20 LOOP
INSERT INTO EXAMPLE1(COL1) VALUES (i);
IF i % 2 = 0 THEN
COMMIT;
ELSE
ROLLBACK;
```

```
END IF;
END LOOP;
END;
/
```

**NOTE**

- COMMIT and ROLLBACK can be used in stored procedures that contain EXCEPTION.
- COMMIT and ROLLBACK can be used in EXCEPTION statements of stored procedures.
- DDL statements after COMMIT or ROLLBACK can be committed or rolled back.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK()
AS
BEGIN
DROP TABLE IF EXISTS TEST_COMMIT;
CREATE TABLE TEST_COMMIT(A INT, B INT);
INSERT INTO TEST_COMMIT SELECT 1, 1;
COMMIT;
CREATE TABLE TEST_ROLLBACK(A INT, B INT);
RAISE EXCEPTION 'RAISE EXCEPTION AFTER COMMIT';
EXCEPTION
WHEN OTHERS THEN
INSERT INTO TEST_COMMIT SELECT 2, 2;
ROLLBACK;
END;
/
```

**NOTE**

A stored procedure that contains COMMIT or ROLLBACK (which means the stored procedure is controlled by BEGIN, START, or END) can be called in a transaction block.

```
gaussdb=# BEGIN;
CALL TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK();
END;
```

**NOTE**

COMMIT and ROLLBACK, as well as IF, FOR, CURSOR LOOP, and WHILE, can be called in most PL/SQL contexts and statements.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TEST_COMMIT2()
IS
BEGIN
DROP TABLE IF EXISTS TEST_COMMIT;
CREATE TABLE TEST_COMMIT(A INT);
FOR I IN REVERSE 3..0 LOOP
INSERT INTO TEST_COMMIT SELECT I;
COMMIT;
END LOOP;
FOR I IN REVERSE 2..4 LOOP
UPDATE TEST_COMMIT SET A=I;
COMMIT;
END LOOP;
EXCEPTION
WHEN OTHERS THEN
INSERT INTO TEST_COMMIT SELECT 4;
COMMIT;
END;
/
```

**NOTE**

GUC parameters in stored procedures can be committed or rolled back.

```
gaussdb=# SHOW explain_perf_mode;
gaussdb=# SHOW enable_force_vector_engine;

gaussdb=# CREATE OR REPLACE PROCEDURE GUC_ROLLBACK()
AS
```

```
BEGIN
 SET enable_force_vector_engine = on;
 COMMIT;
 SET explain_perf_mode TO pretty;
 ROLLBACK;
END;
/

gaussdb=# call GUC_ROLLBACK();
guc_rollback

(1 row)

gaussdb=# SHOW explain_perf_mode;
explain_perf_mode

normal
(1 row)

gaussdb=# SHOW enable_force_vector_engine;
enable_force_vector_engine

on
(1 row)

gaussdb=# SET enable_force_vector_engine = off;
```

** NOTE**

Savepoints can be used in PL/SQL stored procedures to roll back partial transaction modifications.

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE1()
AS
BEGIN
 INSERT INTO EXAMPLE1 VALUES(1);
 SAVEPOINT s1;
 INSERT INTO EXAMPLE1 VALUES(2);
 ROLLBACK TO s1; -- Roll back the insertion of record 2.
 INSERT INTO EXAMPLE1 VALUES(3);
END;
/
```

** NOTE**

You can use a savepoint in a PL/SQL stored procedure to roll back to a savepoint defined outside the stored procedure.

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE2()
AS
BEGIN
 INSERT INTO EXAMPLE1 VALUES(2);
 ROLLBACK TO s1; -- Roll back the insertion of record 2.
 INSERT INTO EXAMPLE1 VALUES(3);
END;
/

gaussdb=# BEGIN;
gaussdb=# INSERT INTO EXAMPLE1 VALUES(1);
gaussdb=# SAVEPOINT s1;
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE2();
stp_savepoint_example2

(1 row)

gaussdb=# SELECT * FROM EXAMPLE1;
col1

```

```

0
4
10
14
16
18
20
3
3
3
2
6
8
12
1
1
(16 rows)

gaussdb=# COMMIT;

```

 **NOTE**

You can use a savepoint defined outside the stored procedure to roll back to a savepoint in a PL/SQL stored procedure.

```

gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
 INSERT INTO EXAMPLE1 VALUES(1);
 SAVEPOINT s1;
 INSERT INTO EXAMPLE1 VALUES(2);
END;
/

gaussdb=# BEGIN;
gaussdb=# INSERT INTO EXAMPLE1 VALUES(3);
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE3();
stp_savepoint_example3

(1 row)

gaussdb=# ROLLBACK TO SAVEPOINT s1; -- Roll back the insertion of record 2 to the stored procedure.
gaussdb=# SELECT * FROM EXAMPLE1;
col1

0
4
10
14
16
18
20
3
3
3
3
2
6
8
12
1
1
1
(18 rows)

gaussdb=# COMMIT;

```

 **NOTE**

The COMMIT and ROLLBACK statements can be called in a function.

```
gaussdb=# CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE1() RETURN INT
AS
EXP INT;
BEGIN
 FOR i IN 0..20 LOOP
 INSERT INTO EXAMPLE1(col1) VALUES (i);
 IF i % 2 = 0 THEN
 COMMIT;
 ELSE
 ROLLBACK;
 END IF;
 END LOOP;
 SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
 RETURN EXP;
END;
/
```

There are the following constraints on the use of COMMIT/ROLLBACK in a stored procedure:

 **NOTE**

A TRIGGER stored procedure cannot contain COMMIT/ROLLBACK or called another stored procedure that contains COMMIT/ROLLBACK.

```
gaussdb=# CREATE OR REPLACE FUNCTION FUNCTION_TRI_EXAMPLE2() RETURN TRIGGER
AS
EXP INT;
BEGIN
 FOR i IN 0..20 LOOP
 INSERT INTO EXAMPLE1(col1) VALUES (i);
 IF i % 2 = 0 THEN
 COMMIT;
 ELSE
 ROLLBACK;
 END IF;
 END LOOP;
 SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
END;
/
```

```
gaussdb=# CREATE TRIGGER TRIGGER_EXAMPLE AFTER DELETE ON EXAMPLE1
FOR EACH ROW EXECUTE PROCEDURE FUNCTION_TRI_EXAMPLE2();
```

```
gaussdb=# DELETE FROM EXAMPLE1;
ERROR: Can not commit/rollback if it's atomic is true: can not use commit rollback in Complex SQL
CONTEXT: PL/pgSQL function function_tri_example2() line 7 at COMMIT
```

 **NOTE**

Stored procedures that contain IMMUTABLE or SHIPPABLE cannot call COMMIT/ROLLBACK or another stored procedure that contains COMMIT/ROLLBACK.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE1()
IMMUTABLE
AS
BEGIN
 FOR i IN 0..20 LOOP
 INSERT INTO EXAMPLE1 (col1) VALUES (i);
 IF i % 2 = 0 THEN
 COMMIT;
 ELSE
 ROLLBACK;
 END IF;
 END LOOP;
```

```
END;
/
gaussdb=# CALL TRANSACTION_EXAMPLE1();
ERROR: Can not commit/rollback if it's atomic is true: commit/rollback/savepoint is not allowed in a non-
volatile function
CONTEXT: PL/pgSQL function transaction_example1() line 7 at COMMIT
```

** NOTE**

Variables declared or imported in stored procedures cannot be committed or rolled back.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE2(EXP_OUT OUT INT)
AS
EXP INT;
BEGIN
 EXP_OUT := 0;
 COMMIT;
 DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
 EXP_OUT := 1;
 ROLLBACK;
 DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
END;
/
```

** NOTE**

Calling in SQL statements (other than Select Procedure) is not supported.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE3()
AS
BEGIN
 FOR i IN 0..20 LOOP
 INSERT INTO EXAMPLE1 (col1) VALUES (i);
 IF i % 2 = 0 THEN
 EXECUTE IMMEDIATE 'COMMIT';
 ELSE
 EXECUTE IMMEDIATE 'ROLLBACK';
 END IF;
 END LOOP;
END;
/
gaussdb=# CALL TRANSACTION_EXAMPLE2(100);
EXP IS:
EXP IS:
exp_out

1
(1 row)
```

** NOTE**

COMMIT/ROLLBACK cannot be called in a stored procedure whose header contains GUC parameters.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE4()
SET ARRAY_NULLS TO "ON"
AS
BEGIN
 FOR i IN 0..20 LOOP
 INSERT INTO EXAMPLE1 (col1) VALUES (i);
 IF i % 2 = 0 THEN
 COMMIT;
 ELSE
 ROLLBACK;
 END IF;
 END LOOP;
END;
/
gaussdb=# CALL TRANSACTION_EXAMPLE4();
ERROR: Can not commit/rollback if it's atomic is true: transaction statement in store procedure with GUC
```

setting in option clause is not supported  
CONTEXT: PL/pgSQL function transaction\_example4() line 6 at COMMIT

**NOTE**

A stored procedure object whose cursor is open cannot contain COMMIT/ROLLBACK.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE5(INTIN IN INT, INTOUT OUT INT)
AS
BEGIN
INTOUT := INTIN + 1;
COMMIT;
END;
/
```

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE6()
AS
CURSOR CURSOR1(EXPIN INT)
IS SELECT TRANSACTION_EXAMPLE5(EXPIN);
INTEXP INT;
BEGIN
FOR i IN 0..20 LOOP
OPEN CURSOR1(i);
FETCH CURSOR1 INTO INTEXP;
INSERT INTO EXAMPLE1(COL1) VALUES (INTEXP);
IF i % 2 = 0 THEN
COMMIT;
ELSE
ROLLBACK;
END IF;
CLOSE CURSOR1;
END LOOP;
END;
/
```

```
gaussdb=# CALL TRANSACTION_EXAMPLE6();
ERROR: Can not commit/rollback if it's atomic is true: transaction statement in store procedure used as
cursor is not supported
CONTEXT: PL/pgSQL function transaction_example5(integer) line 4 at COMMIT
referenced column: transaction_example5
PL/pgSQL function transaction_example6() line 8 at FETCH
```

**NOTE**

COMMIT or ROLLBACK cannot be called in expressions or CURSOR and EXECUTE statements.

```
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func1()
AS
BEGIN
CREATE TABLE TEST_exec(A INT);
COMMIT;
END;
/
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func2()
AS
BEGIN
EXECUTE exec_func1();
COMMIT;
END;
/
gaussdb=# CALL exec_func2();
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CONTEXT: SQL statement "CREATE TABLE TEST_exec(A INT)"
PL/pgSQL function exec_func1() line 3 at SQL statement
PL/pgSQL function exec_func2() line 3 at EXECUTE statement
ERROR: Can not commit/rollback if it's atomic is true: transaction statement in store procedure used as a
expression is not supported
CONTEXT: PL/pgSQL function exec_func1() line 4 at COMMIT
PL/pgSQL function exec_func2() line 3 at EXECUTE statement
```

 **NOTE**

Return values and expression calculation of stored procedures are not supported.

```
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func3(RET_NUM OUT INT)
AS
BEGIN
 RET_NUM := 1+1;
COMMIT;
END;
/
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func4(ADD_NUM IN INT)
AS
SUM_NUM INT;
BEGIN
SUM_NUM := ADD_NUM + exec_func3();
COMMIT;
END;
/
gaussdb=# CALL exec_func4(1);
ERROR: Can not commit/rollback if it's atomic is true: transaction statement in store procedure used as a
expression is not supported
CONTEXT: PL/pgSQL function exec_func3() line 4 at COMMIT
PL/pgSQL function exec_func4(integer) line 4 at assignment
```

 **NOTE**

Savepoints defined outside a stored procedure cannot be released in the stored procedure.

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
 INSERT INTO EXAMPLE1 VALUES(2);
 RELEASE SAVEPOINT s1; -- Release the savepoint defined outside the stored procedure.
 INSERT INTO EXAMPLE1 VALUES(3);
END;
/

gaussdb=# BEGIN;
gaussdb=# INSERT INTO EXAMPLE1 VALUES(1);
gaussdb=# SAVEPOINT s1;
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE3();
gaussdb=# COMMIT;
ERROR: cannot release outer savepoint
CONTEXT: PL/pgSQL function stp_savepoint_example3() line 4 at RELEASE SAVEPOINT
```

## 10.9 Other Statements

### 10.9.1 Lock Operations

GaussDB provides multiple lock modes to control concurrent accesses to table data. These modes are used when Multi-Version Concurrency Control (MVCC) cannot give expected behaviors. Alike, most GaussDB commands automatically apply appropriate locks to ensure that called tables are not deleted or modified in an incompatible manner during command execution. For example, when concurrent operations exist, ALTER TABLE cannot be executed on the same table.

For the complete lock operations, see [LOCK](#).

### 10.9.2 Cursor Operations

GaussDB provides cursors as a data buffer for users to store execution results of SQL statements. Each cursor region has a name. Users can use SQL statements to

obtain records one by one from cursors and grant the records to master variables, then being processed further by host languages.

Cursor operations include cursor definition, open, fetch, and close operations.

For the complete example of cursor operations, see [Explicit Cursor](#).

## 10.10 Cursors

### 10.10.1 Overview

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

#### NOTICE

- If JDBC is used to call a stored procedure whose returned value is a cursor, the returned cursor cannot be used.
- In distributed mode, if the parameter of the stored procedure called in the SQL statement is of the refcursor type or the return value is of the refcursor type, pushdown is not supported.
- When a stored procedure contains COMMIT/ROLLBACK, an explicit cursor caches all data of the cursor to ensure that the cursor is still available after COMMIT/ROLLBACK. If the cursor data volume is large, this process may take a long time.

Cursors are classified into explicit cursors and implicit cursors. [Table 10-2](#) shows the usage conditions of explicit and implicit cursors for different SQL statements.

**Table 10-2** Cursor usage conditions

| SQL Statement                             | Cursor               |
|-------------------------------------------|----------------------|
| Non-query statements                      | Implicit             |
| Query statements with single-line results | Implicit or explicit |
| Query statements with multi-line results  | Explicit             |

### 10.10.2 Explicit Cursor

An explicit cursor is used to process query statements, particularly when query results are multiple records.

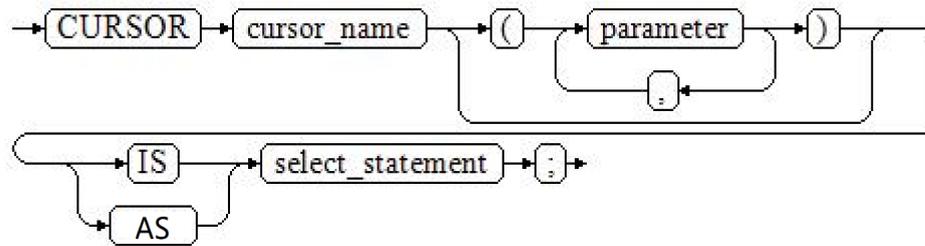
## Procedure

An explicit cursor performs the following six PL/SQL steps to process query statements:

- Step 1** Define a static cursor: Define a cursor name and its corresponding **SELECT** statement.

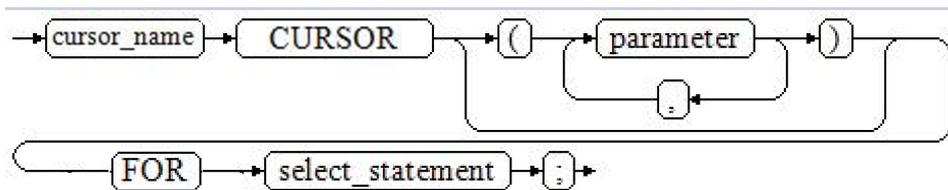
**Figure 10-25** shows the syntax diagram for defining a static cursor.

**Figure 10-25** static\_cursor\_define::=



Or

**Figure 10-26** static\_cursor\_define::=



Parameter description:

- *cursor\_name*: defines a cursor name.
- *parameter*: specifies cursor parameters. Only input parameters are allowed. Its format is as follows:  
parameter\_name datatype
- *select\_statement*: specifies a query statement.

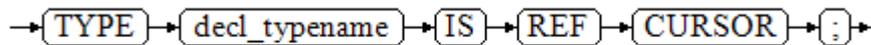
**NOTE**

- The system automatically determines whether the cursor can be used for backward fetching based on the execution plan.
- In syntax, **parameter** can be an output parameter, but its behavior is the same as that of an input parameter.

Define a dynamic cursor: Define a **ref** cursor, which means that the cursor can be opened dynamically by a set of static SQL statements. Define the type of the **ref** cursor first, and then the cursor variable of this cursor type. Dynamically bind a **SELECT** statement through **OPEN FOR** when the cursor is opened.

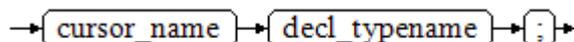
**Figure 10-27** and **Figure 10-28** show the syntax diagrams for defining a dynamic cursor.

Figure 10-27 cursor\_typename::=



GaussDB supports the dynamic cursor type **sys\_refcursor**. A function or stored procedure can use the **sys\_refcursor** parameter to pass on or pass out the cursor result set. A function can return **sys\_refcursor** to return the cursor result set.

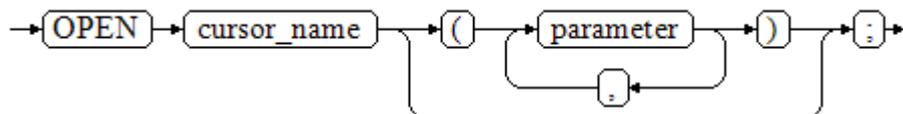
Figure 10-28 dynamic\_cursor\_define::=



**Step 2** Open the static cursor: Execute the **SELECT** statement corresponding to the cursor. The query result is placed in the workspace and the pointer directs to the head of the workspace to identify the cursor result set. If the cursor query statement carries the **FOR UPDATE** option, the **OPEN** statement locks the data rows corresponding to the cursor result set in the database table.

Figure 10-29 shows the syntax diagram for opening a static cursor.

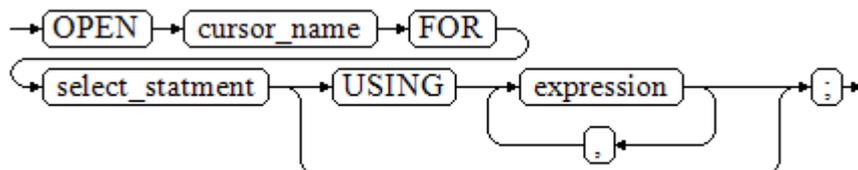
Figure 10-29 open\_static\_cursor::=



Open the dynamic cursor: Use the **OPEN FOR** statement to open the dynamic cursor and the SQL statement is dynamically bound.

Figure 10-30 shows the syntax diagrams for opening a dynamic cursor.

Figure 10-30 open\_dynamic\_cursor::=

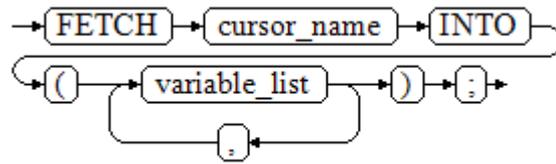


A PL/SQL program cannot use the **OPEN** statement to repeatedly open a cursor.

**Step 3** Fetch cursor data: Retrieve data rows in the result set and place them in specified output variables.

Figure 10-31 shows the syntax diagrams for fetching cursor data.

Figure 10-31 fetch\_cursor::=



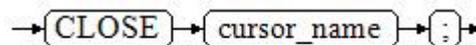
**Step 4** Process the record.

**Step 5** Continue to process until the active set has no record.

**Step 6** Close the cursor: After you fetch and process the data in the cursor result set, close the cursor in time to release system resources used by the cursor and invalidate the workspace of the cursor so that the **FETCH** statement cannot be used to fetch data any more. A closed cursor can be reopened by an **OPEN** statement.

Figure 10-32 shows the syntax diagram for closing a cursor.

Figure 10-32 close\_cursor::=



----End

## Attributes

Cursor attributes are used to control program procedures or know program status. When a DML statement is executed, the PL/SQL opens a built-in cursor and processes its result. A cursor is a memory segment for maintaining query results. It is opened when a DML statement is executed and closed when the execution is finished. An explicit cursor has the following attributes:

- **%FOUND**: Boolean attribute, which returns **TRUE** if the last fetch returns a row.
- **%NOTFOUND**: Boolean attribute, which returns **TRUE** if the last fetch fails to return a row.
- **%ISOPEN**: Boolean attribute, which returns **TRUE** if the cursor has been opened.
- **%ROWCOUNT**: numeric attribute, which returns the number of records fetched from the cursor.

## Examples

DDL and DML statements are prepared. Subsequent examples in this section depend on this case.

```

gaussdb=# drop schema if exists hr cascade;
gaussdb=# create schema hr;
gaussdb=# set current_schema = hr;
gaussdb=# drop table if exists sections;
gaussdb=# drop table if exists staffs;
gaussdb=# drop table if exists department;
-- Create a department table.
gaussdb=# create table sections(

```

```
 section_name varchar(100),
 place_id int,
 section_id int
);
gaussdb=# insert into sections values ('hr',1,1);

-- Create an employee table.
gaussdb=# create table staffs(
 staff_id number(6),
 salary number(8,2),
 section_id int,
 first_name varchar(20)
);
gaussdb=# insert into staffs values (1,100,1,'Tom');

-- Create a department table.
gaussdb=# create table department(
 section_id int
);
-- Specify the method for passing cursor parameters.
CREATE OR REPLACE PROCEDURE cursor_proc1()
AS
DECLARE
 DEPT_NAME VARCHAR(100);
 DEPT_LOC NUMBER(4);
 -- Define a cursor.
 CURSOR C1 IS
 SELECT section_name, place_id FROM hr.sections WHERE section_id <= 50;
 CURSOR C2(sect_id INTEGER) IS
 SELECT section_name, place_id FROM hr.sections WHERE section_id <= sect_id;
 TYPE CURSOR_TYPE IS REF CURSOR;
 C3 CURSOR_TYPE;
 SQL_STR VARCHAR(100);
BEGIN
 OPEN C1;-- Open the cursor.
 LOOP
 -- Fetch data from the cursor.
 FETCH C1 INTO DEPT_NAME, DEPT_LOC;
 EXIT WHEN C1%NOTFOUND;
 DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
 END LOOP;
 CLOSE C1;-- Close the cursor.

 OPEN C2(10);
 LOOP
 FETCH C2 INTO DEPT_NAME, DEPT_LOC;
 EXIT WHEN C2%NOTFOUND;
 DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
 END LOOP;
 CLOSE C2;

 SQL_STR := 'SELECT section_name, place_id FROM hr.sections WHERE section_id <= :DEPT_NO;';
 OPEN C3 FOR SQL_STR USING 50;
 LOOP
 FETCH C3 INTO DEPT_NAME, DEPT_LOC;
 EXIT WHEN C3%NOTFOUND;
 DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
 END LOOP;
 CLOSE C3;
END;
/
CREATE PROCEDURE
CALL cursor_proc1();

hr---1
hr---1
hr---1
cursor_proc1
```

```

(1 row)

DROP PROCEDURE cursor_proc1;
DROP PROCEDURE
-- Give a salary raise to employees whose salary is lower than 3000 by adding 500.
CREATE TABLE hr.staffs_t1 AS TABLE hr.staffs;
INSERT 0 1
CREATE OR REPLACE PROCEDURE cursor_proc2()
AS
DECLARE
 V_EMPNO NUMBER(6);
 V_SAL NUMBER(8,2);
 CURSOR C IS SELECT staff_id, salary FROM hr.staffs_t1;
BEGIN
 OPEN C;
 LOOP
 FETCH C INTO V_EMPNO, V_SAL;
 EXIT WHEN C%NOTFOUND;
 IF V_SAL<=3000 THEN
 UPDATE hr.staffs_t1 SET salary =salary + 500 WHERE staff_id = V_EMPNO;
 END IF;
 END LOOP;
 CLOSE C;
END;
/
CREATE PROCEDURE
CALL cursor_proc2();
cursor_proc2

(1 row)

-- Delete the stored procedure.
DROP PROCEDURE cursor_proc2;
DROP PROCEDURE
DROP TABLE hr.staffs_t1;
DROP TABLE
-- Use function parameters of the SYS_REFCURSOR type.
CREATE OR REPLACE PROCEDURE proc_sys_ref(O OUT SYS_REFCURSOR)
IS
C1 SYS_REFCURSOR;
BEGIN
OPEN C1 FOR SELECT section_ID FROM HR.sections ORDER BY section_ID;
O := C1;
END;
/

DECLARE
C1 SYS_REFCURSOR;
TEMP NUMBER(4);
BEGIN
proc_sys_ref(C1);
LOOP
 FETCH C1 INTO TEMP;
 DBE_OUTPUT.PRINT_LINE(C1%ROWCOUNT);
 EXIT WHEN C1%NOTFOUND;
END LOOP;
END;
/
1
1
ANONYMOUS BLOCK EXECUTE
-- Delete the stored procedure.
DROP PROCEDURE proc_sys_ref;
```

## 10.10.3 Implicit Cursor

Implicit cursors are automatically set by the system for non-query statements such as modify or delete operations, along with their workspace. Implicit cursors are named **SQL**, which is defined by the system.

### Overview

Implicit cursor operations, such as definition, open, value-grant, and close operations, are automatically performed by the system and do not need users to process. Users can use only attributes related to implicit cursors to complete operations. In workspace of implicit cursors, the data of the latest SQL statement is stored and is not related to explicit cursors defined by users.

Format call: **SQL%**

#### NOTE

INSERT, UPDATE, DELETE, and SELECT statements do not need defined cursors.

### Attributes

An implicit cursor has the following attributes:

- **SQL%FOUND**: Boolean attribute, which returns **TRUE** if the last fetch returns a row.
- **SQL%NOTFOUND**: Boolean attribute, which returns **TRUE** if the last fetch fails to return a row.
- **SQL%ROWCOUNT**: numeric attribute, which returns the number of records fetched from the cursor.
- **SQL%ISOPEN**: Boolean attribute, whose value is always **FALSE**. Close implicit cursors immediately after an SQL statement is run.

### Examples

```
-- Delete all employees in a department from the EMP table. If the department has no employees, delete
the department from the DEPT table.
CREATE TABLE hr.staffs_t1 AS TABLE hr.staffs;
CREATE TABLE hr.sections_t1 AS TABLE hr.sections;

CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
 DECLARE
 V_DEPTNO NUMBER(4) := 100;
 BEGIN
 DELETE FROM hr.staffs WHERE section_ID = V_DEPTNO;
 -- Proceed based on cursor status.
 IF SQL%NOTFOUND THEN
 DELETE FROM hr.sections_t1 WHERE section_ID = V_DEPTNO;
 END IF;
 END;
/

CALL proc_cursor3();
proc_cursor3

(1 row)
```

```
-- Drop the stored procedure and the temporary table.
DROP PROCEDURE proc_cursor3;
DROP TABLE hr.staffs_t1;
DROP TABLE hr.sections_t1;
```

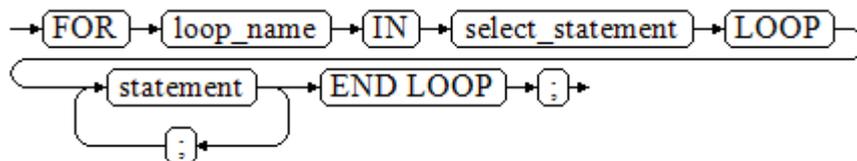
## 10.10.4 Cursor Loop

Use of cursors in WHILE and LOOP statements is called a cursor loop. Generally, OPEN, FETCH, and CLOSE statements are called in this kind of loop. The following describes a loop that simplifies a cursor loop without the need for these operations. This mode is applicable to a static cursor loop, without executing four steps about a static cursor.

### Syntax

Figure 10-33 shows the syntax diagram of the FOR AS loop.

Figure 10-33 FOR\_AS\_loop::=



### Precautions

- The UPDATE operation for the queried table is not allowed in the loop statement.
- The variable *loop\_name* is automatically defined and is valid only in this loop. Its type is the same as that in the query result of *select\_statement*. The value of *loop\_name* is the query result of *select\_statement*.
- The specific type of the **loop\_name** variable is not parsed during compilation. If the specific type needs to be parsed (for example, **loop\_name** is used as the input and output parameters of an overloaded function or stored procedure), a compilation error is reported.
- The **%FOUND**, **%NOTFOUND**, and **%ROWCOUNT** attributes access the same internal variable in GaussDB. Transactions and the anonymous block do not support multiple cursor accesses at the same time.

### Examples

```
BEGIN
FOR ROW_TRANS IN
 SELECT first_name FROM hr.staffs
 LOOP
 DBE_OUTPUT.PRINT_LINE (ROW_TRANS.first_name);
 END LOOP;
END;
/
Tom
ANONYMOUS BLOCK EXECUTE
-- Create a table.
CREATE TABLE integerTable1(A INTEGER) DISTRIBUTE BY hash(A);
CREATE TABLE integerTable2(B INTEGER) DISTRIBUTE BY hash(B);
INSERT INTO integerTable2 VALUES(2);
```

```

-- Multiple cursors share the parameters of cursor attributes.
DECLARE
 CURSOR C1 IS SELECT A FROM integerTable1;-- Declare the cursor.
 CURSOR C2 IS SELECT B FROM integerTable2;
 PI_A INTEGER;
 PI_B INTEGER;
BEGIN
 OPEN C1;-- Open the cursor.
 OPEN C2;
 FETCH C1 INTO PI_A; ---- The values of C1%FOUND and C2%FOUND are FALSE.
 FETCH C2 INTO PI_B; ---- The values of C1%FOUND and C2%FOUND are TRUE.
 -- Determine the cursor status.
 IF C1%FOUND THEN
 IF C2%FOUND THEN
 DBE_OUTPUT.PRINT_LINE('Dual cursor share parameter.');
```

## 10.11 Advanced Packages

Advanced packages have two sets of interfaces. The first set is basic interfaces, and the second set is secondary encapsulation interfaces that are used improve usability. The second set is recommended.

### 10.11.1 Basic APIs

#### 10.11.1.1 PKG\_SERVICE

[Table 10-3](#) lists all APIs supported by the **PKG\_SERVICE** package.

**Table 10-3** PKG\_SERVICE

| API                                                | Description                                                                             |
|----------------------------------------------------|-----------------------------------------------------------------------------------------|
| <a href="#">PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE</a>  | Checks whether a context is registered.                                                 |
| <a href="#">PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS</a> | Deregisters all registered contexts.                                                    |
| <a href="#">PKG_SERVICE.SQL_REGISTER_CONTEXT</a>   | Registers a context.                                                                    |
| <a href="#">PKG_SERVICE.SQL_UNREGISTER_CONTEXT</a> | Deregisters a context.                                                                  |
| <a href="#">PKG_SERVICE.SQL_SET_SQL</a>            | Sets an SQL statement for a context. Currently, only the SELECT statement is supported. |

| API                                                 | Description                                                                                                                      |
|-----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">PKG_SERVICE.SQL_RUN</a>                 | Executes the configured SQL statement on a context.                                                                              |
| <a href="#">PKG_SERVICE.SQL_NEXT_ROW</a>            | Reads the next row of data in a context.                                                                                         |
| <a href="#">PKG_SERVICE.SQL_GET_VALUE</a>           | Reads a dynamically defined column value in a context.                                                                           |
| <a href="#">PKG_SERVICE.SQL_SET_RESULT_TYPE</a>     | Dynamically defines a column of a context based on the type OID.                                                                 |
| <a href="#">PKG_SERVICE.JOB_CANCEL</a>              | Removes a scheduled task by task ID.                                                                                             |
| <a href="#">PKG_SERVICE.JOB_FINISH</a>              | Disables or enables scheduled task execution.                                                                                    |
| <a href="#">PKG_SERVICE.JOB_SUBMIT</a>              | Submits a scheduled task. Job ID can be automatically generated by the system or specified manually.                             |
| <a href="#">PKG_SERVICE.JOB_UPDATE</a>              | Modifies user-definable attributes of a scheduled task, including the task content, next-execution time, and execution interval. |
| <a href="#">PKG_SERVICE.SUBMIT_ON_NODES</a>         | Submits a task to all nodes. The task ID is automatically generated by the system.                                               |
| <a href="#">PKG_SERVICE.ISUBMIT_ON_NODES</a>        | Submits a job to all nodes. The job ID is specified by the user.                                                                 |
| <a href="#">PKG_SERVICE.SQL_GET_ARRAY_RESULT</a>    | Obtains the array value returned in the context.                                                                                 |
| <a href="#">PKG_SERVICE.SQL_GET_VARIABLE_RESULT</a> | Obtains the column value returned in the context.                                                                                |

- PKG\_SERVICE.SQL\_IS\_CONTEXT\_ACTIVE**  
 This function checks whether a context is registered. This function transfers the ID of the context to be queried. If the context exists, **TRUE** is returned. Otherwise, **FALSE** is returned.

The prototype of the PKG\_SERVICE.SQL\_IS\_CONTEXT\_ACTIVE function is as follows:

```
PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE(
context_id IN INTEGER
)
RETURN BOOLEAN;
```

**Table 10-4** PKG\_SERVICE.SQL\_IS\_CONTEXT\_ACTIVE parameter

| Parameter  | Description                      |
|------------|----------------------------------|
| context_id | ID of the context to be queried. |

- PKG\_SERVICE.SQL\_CLEAN\_ALL\_CONTEXTS

This function cancels all contexts.

The prototype of the PKG\_SERVICE.SQL\_CLEAN\_ALL\_CONTEXTS function is as follows:

```
PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS(
)
RETURN VOID;
```

- PKG\_SERVICE.SQL\_REGISTER\_CONTEXT

This function opens a context, which is the prerequisite for the subsequent operations in the context. This function does not transfer any parameter. It automatically generates context IDs in an ascending order and returns values to integer variables.

The prototype of the PKG\_SERVICE.SQL\_REGISTER\_CONTEXT function is as follows:

```
DBE_SQL.REGISTER_CONTEXT(
)
RETURN INTEGER;
```

- PKG\_SERVICE.SQL\_UNREGISTER\_CONTEXT

This function closes a context, which is the end of each operation in the context. If this function is not called when the stored procedure ends, the memory is still occupied by the context. Therefore, remember to close a context when you do not need to use it. If an exception occurs, the stored procedure exits but the context is not closed. Therefore, you are advised to include this API in the exception handling of the stored procedure.

The prototype of the PKG\_SERVICE.SQL\_UNREGISTER\_CONTEXT function is as follows:

```
PKG_SERVICE.SQL_UNREGISTER_CONTEXT(
context_id IN INTEGER
)
RETURN INTEGER;
```

**Table 10-5** PKG\_SERVICE.SQL\_UNREGISTER\_CONTEXT API parameter

| Parameter  | Description                     |
|------------|---------------------------------|
| context_id | ID of the context to be closed. |

- PKG\_SERVICE.SQL\_SET\_SQL

This function parses the query statement of a given context. The input query statement is executed immediately. Currently, only the **SELECT** query statement can be parsed. The statement parameters can be transferred only through the **TEXT** type. The length cannot exceed 1 GB.

The prototype of the PKG\_SERVICE.SQL\_SET\_SQL function is as follows:

```
PKG_SERVICE.SQL_SET_SQL(
context_id IN INTEGER,
```

```
query_string IN TEXT,
language_flag IN INTEGER
)
RETURN BOOLEAN;
```

**Table 10-6** PKG\_SERVICE.SQL\_SET\_SQL API parameter

| Parameter     | Description                                                     |
|---------------|-----------------------------------------------------------------|
| context_id    | ID of the context whose query statement is to be parsed.        |
| query_string  | Query statement to be parsed.                                   |
| language_flag | Version language number. Currently, only <b>1</b> is supported. |

- PKG\_SERVICE.SQL\_RUN

This function executes a given context. It receives a context ID first, and the data obtained after execution is used for subsequent operations. Currently, only the **SELECT** query statement can be executed.

The prototype of the PKG\_SERVICE.SQL\_RUN function is as follows:

```
PKG_SERVICE.SQL_RUN(
context_id IN INTEGER,
)
RETURN INTEGER;
```

**Table 10-7** PKG\_SERVICE.SQL\_RUN API parameters

| Parameter  | Description                                              |
|------------|----------------------------------------------------------|
| context_id | ID of the context whose query statement is to be parsed. |

- PKG\_SERVICE.SQL\_NEXT\_ROW

This function returns the number of data rows returned after the SQL statement is executed. Each time the API is executed, the system obtains a set of new rows until all data is read.

The prototype of the PKG\_SERVICE.SQL\_NEXT\_ROW function is as follows:

```
PKG_SERVICE.SQL_NEXT_ROW(
context_id IN INTEGER,
)
RETURN INTEGER;
```

**Table 10-8** PKG\_SERVICE.SQL\_NEXT\_ROW API parameters

| Parameter  | Description                       |
|------------|-----------------------------------|
| context_id | ID of the context to be executed. |

- PKG\_SERVICE.SQL\_GET\_VALUE

This function returns the context element value in a specified position of a context and accesses the data obtained by PKG\_SERVICE.SQL\_NEXT\_ROW.

The prototype of the PKG\_SERVICE.SQL\_GET\_VALUE function is as follows:

```
PKG_SERVICE.SQL_GET_VALUE(
context_id IN INTEGER,
pos IN INTEGER,
col_type IN ANYELEMENT
)
RETURN ANYELEMENT;
```

**Table 10-9** PKG\_SERVICE.SQL\_GET\_VALUE API parameters

| Parameter  | Description                                                           |
|------------|-----------------------------------------------------------------------|
| context_id | ID of the context to be executed.                                     |
| pos        | Position of a dynamically defined column in the query.                |
| col_type   | Variable of any type, which defines the return value type of columns. |

- PKG\_SERVICE.SQL\_SET\_RESULT\_TYPE

This function defines columns returned from a given context and can be used only for contexts defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The prototype of

PKG\_SERVICE.SQL\_SET\_RESULT\_TYPE is as follows:

```
PKG_SERVICE.SQL_SET_RESULT_TYPE(
context_id IN INTEGER,
pos IN INTEGER,
coltype_oid IN ANYELEMENT,
maxsize IN INTEGER
)
RETURN INTEGER;
```

**Table 10-10** PKG\_SERVICE.SQL\_SET\_RESULT\_TYPE API parameters

| Parameter   | Description                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------|
| context_id  | ID of the context to be executed.                                                                   |
| pos         | Position of a dynamically defined column in the query.                                              |
| coltype_oid | Variable of any type. The OID of the corresponding type can be obtained based on the variable type. |
| maxsize     | Length of a defined column.                                                                         |

- PKG\_SERVICE.JOB\_CANCEL

The stored procedure CANCEL deletes a specified task.

The prototype of the PKG\_SERVICE.JOB\_CANCEL function is as follows:

```
PKG_SERVICE.JOB_CANCEL(
id IN INTEGER);
```

**Table 10-11** PKG\_SERVICE.JOB\_CANCEL API parameter

| Parameter | Type    | Input/Output Parameter | Empty or Not | Description           |
|-----------|---------|------------------------|--------------|-----------------------|
| id        | integer | IN                     | No           | Specifies the job ID. |

- PKG\_SERVICE.JOB\_FINISH

The stored procedure FINISH disables or enables a scheduled task.

The prototype of the PKG\_SERVICE.JOB\_FINISH function is as follows:

```
PKG_SERVICE.JOB_FINISH(
id IN INTEGER,
broken IN BOOLEAN,
next_time IN TIMESTAMP DEFAULT sysdate);
```

**Table 10-12** PKG\_SERVICE.JOB\_FINISH API parameters

| Parameter | Type      | Input/Output Parameter | Empty or Not | Description                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|-----------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id        | integer   | IN                     | No           | Specifies the job ID.                                                                                                                                                                                                                                                                                                                                                                                                          |
| broken    | boolean   | IN                     | No           | Specifies the status flag, <b>true</b> for broken and <b>false</b> for not broken. The current job is updated based on the parameter value <b>true</b> or <b>false</b> . If the parameter is left empty, the job status remains unchanged.                                                                                                                                                                                     |
| next_time | timestamp | IN                     | Yes          | Specifies the next execution time. The default value is the current system time. If <b>broken</b> is set to <b>true</b> , <b>next_time</b> is updated to '4000-1-1'. If <b>broken</b> is set to <b>false</b> and <b>next_time</b> is not empty, <b>next_time</b> is updated for the job. If <b>next_time</b> is empty, it will not be updated. This parameter can be omitted, and its default value will be used in this case. |

- PKG\_SERVICE.JOB\_SUBMIT

The stored procedure JOB\_SUBMIT commits a scheduled task provided by the system.

The prototype of the PKG\_SERVICE.JOB\_SUBMIT function is as follows:

```
PKG_SERVICE.JOB_SUBMIT(
id IN BIGINT,
content IN TEXT,
```

```
next_date IN TIMESTAMP DEFAULT sysdate,
interval_time IN TEXT DEFAULT 'null',
job OUT INTEGER);
```

 **NOTE**

When a scheduled task (using **JOB**) is created, the system binds the current database and the username to the task by default. This function can be called by using **call** or **select**. If you call this function by using **select**, there is no need to specify output parameters. To call this function within a stored procedure, use **perform**. If the committed SQL statement task uses a non-public schema, specify the schema to a table schema or a function schema, or add **set current\_schema = xxx** before the SQL statement.

**Table 10-13** PKG\_SERVICE.JOB\_SUBMIT API parameters

| Parameter     | Type      | Input/Output Parameter | Empty or Not | Description                                                                                                                                                                                                                                                                                                     |
|---------------|-----------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id            | bigint    | IN                     | No           | Specifies the job ID. If the input ID is <b>NULL</b> , a job ID is generated internally.                                                                                                                                                                                                                        |
| content       | text      | IN                     | No           | Specifies the SQL statement to be executed. One or multiple DML statements, anonymous blocks, and statements for calling stored procedures, or all three combined are supported.                                                                                                                                |
| next_time     | timestamp | IN                     | No           | Specifies the next time the job will be executed. The default value is the current system time ( <b>sysdate</b> ). If the specified time has past, the job is executed at the time it is committed.                                                                                                             |
| interval_time | text      | IN                     | Yes          | Calculates the next time to execute the job. It can be an interval expression, or <b>sysdate</b> followed by a numeric value, for example, <b>sysdate+1.0/24</b> . If this parameter is left empty or set to <b>null</b> , the job will be executed only once, and the job status will change to 'd' afterward. |
| job           | integer   | OUT                    | No           | Specifies the job ID. The value ranges from 1 to 32767. When <b>pkg_service.job_submit</b> is called using <b>select</b> , this parameter can be omitted.                                                                                                                                                       |

Example:

```
CREATE TABLE test_table(a int);
CREATE TABLE
```

```

CREATE OR REPLACE PROCEDURE test_job(a in int) IS
BEGIN
INSERT INTO test_table VALUES(a);
COMMIT;
END;
/
CREATE PROCEDURE
SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call pro_xxx()';, to_date('20180101','yyyymmdd'),'sysdate
+1');
job_submit

 28269
(1 row)
SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call pro_xxx()';, to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
job_submit

 1506
(1 row)

CALL PKG_SERVICE.JOB_SUBMIT(NULL, 'INSERT INTO T_JOB VALUES(1); call pro_1(); call pro_2()';,
add_months(to_date('201701','yyyymm'),1), 'date_trunc("day",SYSDATE) + 1 +(8*60+30.0)/
(24*60)' ,,jobid);
job

 14131
(1 row)

SELECT PKG_SERVICE.JOB_SUBMIT (101, 'insert_msg_statistic1';, sysdate, 'sysdate+3.0/24');
job_submit

 101
(1 row)

```

- **PKG\_SERVICE.JOB\_UPDATE**

The stored procedure UPDATE modifies user-definable attributes of a task, including the task content, next-execution time, and execution interval.

The prototype of the PKG\_SERVICE.JOB\_UPDATE function is as follows:

```

PKG_SERVICE.JOB_UPDATE(
id IN BIGINT,
next_time IN TIMESTAMP,
interval_time IN TEXT,
content IN TEXT);

```

**Table 10-14** PKG\_SERVICE.JOB\_UPDATE API parameters

| Parameter | Type    | Input/Output Parameter | Empty or Not | Description           |
|-----------|---------|------------------------|--------------|-----------------------|
| id        | integer | IN                     | No           | Specifies the job ID. |

| Parameter     | Type      | Input/Output Parameter | Empty or Not | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------|-----------|------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| next_time     | timestamp | IN                     | Yes          | Specifies the next execution time. If this parameter is left empty, the system does not update the <b>next_time</b> parameter for the specified job. Otherwise, the system updates the <b>next_time</b> parameter for the specified job.                                                                                                                                                                                                                          |
| interval_time | text      | IN                     | Yes          | Specifies the time expression for calculating the next time the job will be executed. If this parameter is left empty, the system does not update the <b>interval_time</b> parameter for the specified job. Otherwise, the system updates the <b>interval_time</b> parameter for the specified job after necessary validity check. If this parameter is set to <b>null</b> , the job will be executed only once, and the job status will change to 'd' afterward. |
| content       | text      | IN                     | Yes          | Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left empty, the system does not update the <b>content</b> parameter for the specified job. Otherwise, the system updates the <b>content</b> parameter for the specified job.                                                                                                                                                                             |

Example:

```
CALL PKG_SERVICE.JOB_UPDATE(101, sysdate, 'sysdate + 1.0/1440', 'call userproc();');
job_update

(1 row)
CALL PKG_SERVICE.JOB_UPDATE(101, sysdate, 'sysdate + 1.0/1440', 'insert into tbl_a values(sysdate);');
job_update

(1 row)
```

- PKG\_SERVICE.SUBMIT\_ON\_NODES

The stored procedure SUBMIT\_ON\_NODES creates a scheduled task on all CNs and DNs. Only SYSADMIN and MONADMIN have this permission.

The prototype of the PKG\_SERVICE.SUBMIT\_ON\_NODES function is as follows:

```
PKG_SERVICE.SUBMIT_ON_NODES(
node_name IN NAME,
database IN NAME,
what IN TEXT,
```

```
next_date IN TIMESTAMP WITHOUT TIME ZONE,
job_interval IN TEXT,
job OUT INTEGER);
```

**Table 10-15** PKG\_SERVICE.SUBMIT\_ON\_NODES API parameters

| Parameter    | Type      | Input/Output Parameter | Empty or Not | Description                                                                                                                                                                                                                                                                                                            |
|--------------|-----------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| node_name    | text      | IN                     | No           | Specifies the job execution node. Currently, the value can only be <b>'ALL_NODE'</b> (indicating that the job is executed on all nodes) or <b>'CCN'</b> (indicating that the job is executed on the central coordinator node).                                                                                         |
| database     | text      | IN                     | No           | Database used by a cluster job. When the node type is <b>ALL_NODE</b> , the value can only be <b>postgres</b> .                                                                                                                                                                                                        |
| what         | text      | IN                     | No           | Specifies the SQL statement to be executed. One or multiple DML statements, anonymous blocks, and statements for calling stored procedures, or all three combined are supported.                                                                                                                                       |
| nextdate     | timestamp | IN                     | No           | Specifies the next time the job will be executed. The default value is the current system time ( <b>sysdate</b> ). If the specified time has past, the job is executed at the time it is committed.                                                                                                                    |
| job_interval | text      | IN                     | No           | Calculates the next time to execute the job. It can be an interval expression, or <b>sysdate</b> followed by a numeric value, for example, <b>sysdate+1.0/24</b> . If this parameter is left empty or set to <b>null</b> , the job will be executed only once, and the job status will change to <b>'d'</b> afterward. |
| job          | integer   | OUT                    | No           | Specifies the job ID. The value ranges from 1 to 32767. When <code>dbms.submit_on_nodes</code> is called using <code>SELECT</code> , this parameter can be omitted.                                                                                                                                                    |

**Example:**

```
SELECT pkg_service.submit_on_nodes('ALL_NODE', 'postgres', 'select
capture_view_to_json("dbe_perf.statement", 0);', sysdate, 'interval "60 second"');
submit_on_nodes

25376
```

```
(1 row)
SELECT pkg_service.submit_on_nodes('CCN', 'postgres', 'select
capture_view_to_json("dbe_perf.statement", 0);', sysdate, 'interval '60 second"');
submit_on_nodes

 4973
(1 row)
```

- **PKG\_SERVICE.ISUBMIT\_ON\_NODES**  
ISUBMIT\_ON\_NODES has the same syntax function as SUBMIT\_ON\_NODES, but the first parameter of ISUBMIT\_ON\_NODES is an input parameter, that is, a specified task ID. In contrast, that last parameter of ISUBMIT\_ON\_NODES is an output parameter, indicating the task ID automatically generated by the system. Only SYSADMIN and MONADMIN have this permission.
- **PKG\_SERVICE.SQL\_GET\_ARRAY\_RESULT**  
This function is used to return the value of the bound OUT parameter of the array type and obtain the OUT parameter in a stored procedure.

The prototype of the PKG\_SERVICE.SQL\_GET\_ARRAY\_RESULT function is as follows:

```
PKG_SERVICE.SQL_GET_ARRAY_RESULT(
 context_id in int,
 pos in VARCHAR2,
 column_value inout anyarray,
 result_type in anyelement
);
```

**Table 10-16** PKG\_SERVICE.SQL\_GET\_ARRAY\_RESULT parameters

| Parameter    | Description                      |
|--------------|----------------------------------|
| context_id   | ID of the context to be queried. |
| pos          | Name of the bound parameter.     |
| column_value | Return value.                    |
| result_type  | Return type.                     |

- **PKG\_SERVICE.SQL\_GET\_VARIABLE\_RESULT**  
This function is used to return the value of the bound OUT parameter of the non-array type and obtain the OUT parameter in a stored procedure.  
The prototype of the PKG\_SERVICE.SQL\_GET\_VARIABLE\_RESULT function is as follows:

```
PKG_SERVICE.SQL_GET_VARIABLE_RESULT(
 context_id in int,
 pos in VARCHAR2,
 result_type in anyelement
)
RETURNS anyelement;
```

**Table 10-17** PKG\_SERVICE.SQL\_GET\_VARIABLE\_RESULT parameters

| Parameter  | Description                      |
|------------|----------------------------------|
| context_id | ID of the context to be queried. |
| pos        | Name of the bound parameter.     |

| Parameter   | Description  |
|-------------|--------------|
| result_type | Return type. |

### 10.11.1.2 PKG\_UTIL

**Table 10-18** lists all APIs supported by the PKG\_UTIL package.

**Table 10-18** PKG\_UTIL

| API                                  | Description                                                                                                                                          |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>PKG_UTIL.LOB_READ</b>             | Reads a part of a LOB.                                                                                                                               |
| <b>PKG_UTIL.LOB_WRITE</b>            | Writes the source object to the target object in the specified format.                                                                               |
| <b>PKG_UTIL.LOB_APPEND</b>           | Appends the source LOB to the target LOB.                                                                                                            |
| <b>PKG_UTIL.LOB_COMPARE</b>          | Compares two LOBs based on the specified length.                                                                                                     |
| <b>PKG_UTIL.LOB_MATCH</b>            | Returns the position of the <i>M</i> th occurrence of a character string in a LOB.                                                                   |
| <b>PKG_UTIL.LOB_RESET</b>            | Resets the character in specified position of a LOB to a specified character.                                                                        |
| <b>PKG_UTIL.LOB_GET_LENGTH</b>       | Obtains and returns the specified length of a LOB.                                                                                                   |
| <b>PKG_UTIL.LOB_READ_HUGE</b>        | Reads a part of the LOB content based on the specified length and initial position offset, and returns the read LOB and length.                      |
| <b>PKG_UTIL.LOB_WRITEAPPEND_HUGE</b> | Reads the content of a specified length from the source BLOB or CLOB, appends the content to the target BLOB or CLOB, and returns the target object. |
| <b>PKG_UTIL.LOB_APPEND_HUGE</b>      | Appends the source BLOB or CLOB to the target BLOB/CLOB and returns the target object.                                                               |
| <b>PKG_UTIL.READ_BFILE_TO_BLOB</b>   | Loads the source BFILE file to the target BLOB and returns the target object.                                                                        |

| API                                         | Description                                                                                                                                                                                                                                       |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">PKG_UTIL.LOB_COPY_HUGE</a>      | Reads the content of a specified length from the specified offset position of the source BLOB or CLOB, writes the content to the specified offset position of the target BLOB or CLOB, and returns the target object.                             |
| <a href="#">PKG_UTIL.BLOB_RESET</a>         | Sets a segment of data in a BLOB to the specified value and returns the processed BLOB and the actually processed length.                                                                                                                         |
| <a href="#">PKG_UTIL.CLOB_RESET</a>         | Sets a segment of data in a CLOB to spaces and returns the processed CLOB and the actually processed length.                                                                                                                                      |
| <a href="#">PKG_UTIL.LOADBLOBFROMFILE</a>   | Reads the content of a specified length from the specified offset position of the source BFILE object, writes the content to the specified offset position of the target BLOB, and returns the target object, read position, and write position.  |
| <a href="#">PKG_UTIL.LOADCLOBFROMFIL...</a> | Reads the content of a specified length from the specified offset position of the source BFILE object, writes the content to the specified offset position of the target CLOB, and returns the target object, read position, and write position.  |
| <a href="#">PKG_UTIL.LOB_CONVERTTOBL...</a> | Reads the content of a specified length from the specified offset position of the source CLOB, converts the content into a BLOB, and writes the BLOB to the specified position of target LOB. <i>amount</i> indicates the length to be converted. |
| <a href="#">PKG_UTIL.LOB_CONVERTTOCL...</a> | Reads the content of a specified length from the specified offset position of the source CLOB, converts the content into a CLOB, and writes the CLOB to the specified position of target LOB. <i>amount</i> indicates the length to be converted. |
| <a href="#">PKG_UTIL.LOB_RAWTOTEXT</a>      | Converts the raw type to the text type.                                                                                                                                                                                                           |
| <a href="#">PKG_UTIL.BFILE_GET LENGT...</a> | Obtains and returns the specified length of a BFILE file.                                                                                                                                                                                         |

| API                                         | Description                                                                                                                                                                                                 |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">PKG_UTIL.BFILE_OPEN</a>         | Opens a BFILE file and returns its file descriptor.                                                                                                                                                         |
| <a href="#">PKG_UTIL.BFILE_CLOSE</a>        | Closes a BFILE file.                                                                                                                                                                                        |
| <a href="#">PKG_UTIL.LOB_WRITE_HUGE</a>     | Reads the specified length of the source object from the start position, writes the content to the specified offset position of the target LOB, overrides the original content, and returns the target LOB. |
| <a href="#">PKG_UTIL.IO_PRINT</a>           | Displays character strings.                                                                                                                                                                                 |
| <a href="#">PKG_UTIL.RAW_GET_LENGTH</a>     | Obtains the length of RAW data.                                                                                                                                                                             |
| <a href="#">PKG_UTIL.RAW_CAST_FROM_V...</a> | Converts VARCHAR2 data to RAW data.                                                                                                                                                                         |
| <a href="#">PKG_UTIL.RAW_CAST_FROM_....</a> | Converts binary integers to RAW data.                                                                                                                                                                       |
| <a href="#">PKG_UTIL.RAW_CAST_TO_BIN...</a> | Converts RAW data to binary integers.                                                                                                                                                                       |
| <a href="#">PKG_UTIL.RANDOM_SET_SEED</a>    | Sets a random seed.                                                                                                                                                                                         |
| <a href="#">PKG_UTIL.RANDOM_GET_VALU...</a> | Returns a random value.                                                                                                                                                                                     |
| <a href="#">PKG_UTIL.FILE_SET_DIRNAM...</a> | Sets the directory to be operated.                                                                                                                                                                          |
| <a href="#">PKG_UTIL.FILE_OPEN</a>          | Opens a file based on the specified file name and directory.                                                                                                                                                |
| <a href="#">PKG_UTIL.FILE_SET_MAX_LI...</a> | Sets the maximum length of a line to be written to a file.                                                                                                                                                  |
| <a href="#">PKG_UTIL.FILE_IS_CLOSE</a>      | Checks whether a file handle is closed.                                                                                                                                                                     |
| <a href="#">PKG_UTIL.FILE_READ</a>          | Reads data of a specified length from an open file handle.                                                                                                                                                  |
| <a href="#">PKG_UTIL.FILE_READLINE</a>      | Reads a line of data from an open file handle.                                                                                                                                                              |
| <a href="#">PKG_UTIL.FILE_WRITE</a>         | Writes the data specified in the buffer to a file.                                                                                                                                                          |
| <a href="#">PKG_UTIL.FILE_WRITELINE</a>     | Writes the buffer to a file and adds newline characters.                                                                                                                                                    |
| <a href="#">PKG_UTIL.FILE_NEWLINE</a>       | Adds a line.                                                                                                                                                                                                |
| <a href="#">PKG_UTIL.FILE_READ_RAW</a>      | Reads binary data of a specified length from an open file handle.                                                                                                                                           |
| <a href="#">PKG_UTIL.FILE_WRITE_RAW</a>     | Writes binary data to a file.                                                                                                                                                                               |

| API                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">PKG_UTIL.FILE_FLUSH</a>         | Writes data from a file handle to a physical file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <a href="#">PKG_UTIL.FILE_CLOSE</a>         | Closes an open file handle.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <a href="#">PKG_UTIL.FILE_REMOVE</a>        | Deletes a physical file. To do so, you must have the corresponding permission.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <a href="#">PKG_UTIL.FILE_RENAME</a>        | Renames a file on the disk, similar to <b>mv</b> in Unix.                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <a href="#">PKG_UTIL.FILE_SIZE</a>          | Returns the size of a file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <a href="#">PKG_UTIL.FILE_BLOCK_SIZE</a>    | Returns the number of blocks contained in a file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <a href="#">PKG_UTIL.FILE_EXISTS</a>        | Checks whether a file exists.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <a href="#">PKG_UTIL.FILE_GETPOS</a>        | Specifies the offset of a returned file, in bytes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <a href="#">PKG_UTIL.FILE_SEEK</a>          | Sets the offset for file position.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <a href="#">PKG_UTIL.FILE_CLOSE_ALL</a>     | Closes all file handles opened in a session.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <a href="#">PKG_UTIL.EXCEPTION_REPOR...</a> | Throws an exception.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <a href="#">PKG_UTIL.UTILITY_COMPILE...</a> | Recompiles specified schemas, functions, and stored procedures. If an error is reported when compiling a PL/SQL object, the PL/SQL object is returned and the recompilation stops. This package has been deprecated. <code>pkg_util.gs_compile_schema</code> is recommended.                                                                                                                                                                                                                                                          |
| <a href="#">PKG_UTIL.GS_COMPILE_SCH....</a> | Recompiles specified schemas, functions, and stored procedures. If an error is reported when compiling a PL/SQL object, the exception is captured and the recompilation continues to compile other objects until all objects are compiled or the number of recompilation attempts reaches the upper limit.<br><br>When the advanced package is executed through JDBC, the SQLSTATE 00000 is displayed, which indicates that the operation is successful. For details, see "Standard SQL Error Codes" in <i>Error Code Reference</i> . |

- **PKG\_UTIL.LOB\_GET\_LENGTH**

Obtains the length of the input data.

The prototype of the PKG\_UTIL.LOB\_GET\_LENGTH function is as follows:

```
PKG_UTIL.LOB_GET_LENGTH(
lob IN CLOB
)
RETURN INTEGER;

PKG_UTIL.LOB_GET_LENGTH(
lob IN BLOB
)
RETURN INTEGER;
```

**Table 10-19** PKG\_UTIL.LOB\_GET\_LENGTH API parameters

| Parameter | Type          | Input/Output Parameter | Can Be Empty | Description                                          |
|-----------|---------------|------------------------|--------------|------------------------------------------------------|
| lob       | CLOB/<br>BLOB | IN                     | No           | Indicates the object whose length is to be obtained. |

- **PKG\_UTIL.LOB\_READ**

Reads an object and returns the specified part.

The prototype of the PKG\_UTIL.LOB\_READ function is as follows:

```
PKG_UTIL.LOB_READ(
lob IN ANYELEMENT,
len IN INT,
start IN INT,
mode IN INT
)
RETURN ANYELEMENT;
```

**Table 10-20** PKG\_UTIL.LOB\_READ API parameters

| Parameter | Type              | Input/Output Parameter | Can Be Empty | Description                                  |
|-----------|-------------------|------------------------|--------------|----------------------------------------------|
| lob       | CLOB<br>/<br>BLOB | IN                     | No           | Specifies CLOB or BLOB data.                 |
| len       | INT               | IN                     | No           | Specifies the length of the returned result. |
| start     | INT               | IN                     | No           | Specifies the offset to the first character. |

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                        |
|-----------|------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| mode      | INT  | IN                     | No           | Specifies the type of the read operation. <b>0</b> indicates <b>READ</b> , <b>1</b> indicates <b>TRIM</b> , and <b>2</b> indicates <b>SUBSTR</b> . |

- **PKG\_UTIL.LOB\_WRITE**

Writes the source object to the target object based on the specified parameters and returns the target object.

The prototype of the PKG\_UTIL.LOB\_WRITE function is as follows:

```

PKG_UTIL.LOB_WRITE(
dest_lob INOUT BLOB,
src_lob IN RAW
len IN INT,
start_pos IN BIGINT
)
RETURN BLOB;
PKG_UTIL.LOB_WRITE(
dest_lob INOUT CLOB,
src_lob IN VARCHAR2
len IN INT,
start_pos IN BIGINT
)
RETURN CLOB;

```

**Table 10-21** PKG\_UTIL.LOB\_WRITE API parameters

| Parameter | Type        | Input/Output Parameter | Can Be Empty | Description                                               |
|-----------|-------------|------------------------|--------------|-----------------------------------------------------------|
| dest_lob  | CLOB / BLOB | INOUT                  | No           | Specifies the target object that data will be written to. |
| src_lob   | CLOB / BLOB | IN                     | No           | Specifies the source object to be written.                |
| len       | INT         | IN                     | No           | Specifies the write length of the source object.          |
| start_pos | BIGINT      | IN                     | No           | Specifies the write start position of the target object.  |

- **PKG\_UTIL.LOB\_APPEND**

Appends the source object to the target BLOB/CLOB and returns the target BLOB/CLOB.

The prototype of the PKG\_UTIL.LOB\_APPEND function is as follows:

```

PKG_UTIL.LOB_APPEND(
dest_lob INOUT BLOB,
src_lob IN BLOB,
len IN INT DEFAULT NULL
)
RETURN BLOB;

PKG_UTIL.LOB_APPEND(
dest_lob INOUT CLOB,
src_lob IN CLOB,
len IN INT DEFAULT NULL
)
RETURN CLOB;

```

**Table 10-22** PKG\_UTIL.LOB\_APPEND API parameters

| Parameter | Type      | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                          |
|-----------|-----------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_lob  | CLOB/BLOB | INOUT                  | No           | Specifies the target BLOB/CLOB that data will be written to.                                                                                                         |
| src_lob   | CLOB/BLOB | IN                     | No           | Specifies the source BLOB/CLOB to be written.                                                                                                                        |
| len       | INT       | IN                     | Yes          | Length read from <i>src</i> and appended to <i>dest</i> . The default value is null, indicating that all content of <i>src</i> is read and appended to <i>dest</i> . |

- PKG\_UTIL.LOB\_COMPARE

Checks whether objects are the same based on the specified start position and size. If **lob1** is larger, **1** is returned. If **lob2** is larger, **-1** is returned. If **lob1** is equal to **lob2**, **0** is returned.

The prototype of the PKG\_UTIL.LOB\_COMPARE function is as follows:

```

PKG_UTIL.LOB_COMPARE(
lob1 IN ANYELEMENT,
lob2 IN ANYELEMENT,
len IN INT DEFAULT 1073741771,
start_pos1 IN INT DEFAULT 1,
start_pos2 IN INT DEFAULT 1
)
RETURN INTEGER;

```

**Table 10-23** PKG\_UTIL.LOB\_COMPARE API parameters

| Parameter  | Type      | Input / Output Parameter | Can Be Empty | Description                                    |
|------------|-----------|--------------------------|--------------|------------------------------------------------|
| lob1       | CLOB/BLOB | IN                       | No           | Indicates the character string for comparison. |
| lob2       | CLOB/BLOB | IN                       | No           | Indicates the character string for comparison. |
| len        | INT       | IN                       | No           | Indicates the length to be compared.           |
| start_pos1 | INT       | IN                       | No           | Specifies the start offset of <b>lob1</b> .    |
| start_pos2 | INT       | IN                       | No           | Specifies the start offset of <b>lob2</b> .    |

- PKG\_UTIL.LOB\_MATCH

Returns the position where a pattern is displayed in a LOB for the *match\_nth* time.

The prototype of the PKG\_UTIL.LOB\_MATCH function is as follows:

```
PKG_UTIL.LOB_MATCH(
lob IN ANYELEMENT,
pattern IN ANYELEMENT,
start IN INT,
match_nth IN INT DEFAULT 1
)
RETURN INTEGER;
```

**Table 10-24** PKG\_UTIL.LOB\_MATCH API parameters

| Parameter | Type      | Input / Output Parameter | Can Be Empty | Description                                      |
|-----------|-----------|--------------------------|--------------|--------------------------------------------------|
| lob       | CLOB/BLOB | IN                       | No           | Indicates the character string for comparison.   |
| pattern   | CLOB/BLOB | IN                       | No           | Specifies the pattern to be matched.             |
| start     | INT       | IN                       | No           | Specifies the start position for LOB comparison. |

| Parameter | Type | Input / Output Parameter | Can Be Empty | Description                   |
|-----------|------|--------------------------|--------------|-------------------------------|
| match_nth | INT  | IN                       | No           | Specifies the matching times. |

- **PKG\_UTIL.LOB\_RESET**

Clears a character string and resets the string to the value of **value**.

The prototype of the PKG\_UTIL.LOB\_RESET function is as follows:

```
PKG_UTIL.LOB_RESET(
lob IN BLOB,
len IN INT,
start IN INT,
value IN INT DEFAULT 0
)
RETURN RECORD;
```

**Table 10-25** PKG\_UTIL.LOB\_RESET API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                     |
|-----------|------|------------------------|--------------|-------------------------------------------------|
| lob       | BLOB | IN                     | No           | Indicates the character string for reset.       |
| len       | INT  | IN                     | No           | Specifies the length of the string to be reset. |
| start     | INT  | IN                     | No           | Specifies the start position for reset.         |
| value     | INT  | IN                     | Yes          | Sets characters. Default value: <b>0</b> .      |

- **PKG\_UTIL.LOB\_GET\_LENGTH**

Obtains and returns the specified length of a LOB.

The prototype of the PKG\_UTIL.LOB\_GET\_LENGTH function is as follows:

```
PKG_UTIL.LOB_GET_LENGTH(
lob IN BLOB)
RETURN BIGINT;

PKG_UTIL.LOB_GET_LENGTH(
lob IN CLOB)
RETURN BIGINT;
```

**Table 10-26** PKG\_UTIL.LOB\_GET\_LENGTH API parameters

| Parameter | Type      | Input/Output Parameter | Can Be Empty | Description |
|-----------|-----------|------------------------|--------------|-------------|
| lob       | BLOB/CLOB | IN                     | No           | LOB type    |

- PKG\_UTIL.LOB\_READ\_HUGE

Reads a part of the LOB content based on the specified length and initial position offset, and returns the read LOB and length.

The prototype of the PKG\_UTIL.LOB\_READ\_HUGE function is as follows:

```

PKG_UTIL.LOB_READ_HUGE(
 lob IN CLOB,
 len IN BIGINT,
 start_pos IN BIGINT,
 mode IN INTEGER)
RETURN RECORD;

PKG_UTIL.LOB_READ_HUGE(
 lob IN BLOB,
 len IN BIGINT,
 start_pos IN BIGINT,
 mode IN INTEGER)
RETURN RECORD;

PKG_UTIL.LOB_READ_HUGE(
 fd IN INTEGER,
 len IN BIGINT,
 start_pos IN BIGINT,
 mode IN INTEGER)
RETURN RECORD;

```

**Table 10-27** PKG\_UTIL.LOB\_READ\_HUGE API parameters

| Parameter | Type              | Input/Output Parameter | Can Be Empty | Description                                         |
|-----------|-------------------|------------------------|--------------|-----------------------------------------------------|
| lob/fd    | BLOB/CLOB/INTEGER | IN                     | No           | File descriptor of the specified LOB or BFILE file. |
| len       | BIGINT            | IN                     | No           | Length to read.                                     |
| start_pos | BIGINT            | IN                     | No           | Offset position from which read starts.             |

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                              |
|-----------|---------|------------------------|--------------|------------------------------------------|
| mode      | INTEGER | IN                     | No           | Read mode (0: read, 1: trim, 2: substr). |

- **PKG\_UTIL.LOB\_WRITEAPPEND\_HUGE**

Reads the content of a specified length from the source BLOB or CLOB, appends the content to the target BLOB or CLOB, and returns the target object.

The prototype of the PKG\_UTIL.LOB\_WRITEAPPEND\_HUGE function is as follows:

```
PKG_UTIL.LOB_WRITEAPPEND_HUGE(
 dest_lob INOUT CLOB,
 len IN INTEGER,
 src_lob IN VARCHAR2
)RETURN CLOB;

PKG_UTIL.LOB_WRITEAPPEND_HUGE(
 dest_lob INOUT BLOB,
 len IN INTEGER,
 src_lob IN RAW
)RETURN BLOB;
```

**Table 10-28** PKG\_UTIL.LOB\_WRITEAPPEND\_HUGE API parameters

| Parameter | Type         | Input/Output Parameter | Can Be Empty | Description                                                                                                              |
|-----------|--------------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------|
| dest_lob  | BLOB/CLOB    | INOUT                  | No           | Target BLOB/CLOB to which data is written.                                                                               |
| len       | INTEGER      | IN                     | Yes          | Length of the source object to be written. If the value is <b>NULL</b> , the entire source object is written by default. |
| src_lob   | VARCHAR2/RAW | IN                     | No           | BLOB/CLOB from which data is to be written.                                                                              |

- **PKG\_UTIL.LOB\_APPEND\_HUGE**

Appends the source BLOB or CLOB to the target BLOB/CLOB and returns the target object.

The prototype of the PKG\_UTIL.LOB\_APPEND\_HUGE function is as follows:

```

PKG_UTIL.LOB_APPEND_HUGE(
 dest_lob INOUT BLOB,
 src_lob IN BLOB)
RETURN BLOB;
PKG_UTIL.LOB_APPEND_HUGE(
 dest_lob INOUT CLOB,
 src_lob IN CLOB)
RETURN CLOB;

```

**Table 10-29** PKG\_UTIL.LOB\_APPEND\_HUGE API parameters

| Parameter | Type      | Input/Output Parameter | Can Be Empty | Description                                 |
|-----------|-----------|------------------------|--------------|---------------------------------------------|
| dest_lob  | BLOB/CLOB | INOUT                  | No           | Target BLOB/CLOB to which data is written.  |
| src_lob   | BLOB/CLOB | IN                     | No           | BLOB/CLOB from which data is to be written. |

- **PKG\_UTIL.READ\_BFILE\_TO\_BLOB**

Loads the source BFILE file to the target BLOB and returns the target object.

The prototype of the PKG\_UTIL.READ\_BFILE\_TO\_BLOB function is as follows:

```

PKG_UTIL.READ_BFILE_TO_BLOB(
 fd IN INTEGER
)RETURN BLOB;

```

**Table 10-30** PKG\_UTIL.READ\_BFILE\_TO\_BLOB API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                   |
|-----------|---------|------------------------|--------------|-------------------------------|
| fd        | INTEGER | IN                     | No           | Source BFILE file to be read. |

- **PKG\_UTIL.LOB\_COPY\_HUGE**

Reads the content of a specified length from the specified offset position of the source BLOB or CLOB, writes the content to the specified offset position of the target BLOB or CLOB, and returns the target object.

The prototype of the PKG\_UTIL.LOB\_COPY\_HUGE function is as follows:

```

PKG_UTIL.LOB_COPY_HUGE(
 lob_obj INOUT BLOB,
 source_obj IN BLOB,
 amount IN BIGINT,
 dest_offset IN BIGINT DEFAULT 1,
 src_offset IN BIGINT DEFAULT 1
)

```

```

)RETURN BLOB;

PKG_UTIL.LOB_COPY_HUGE(
 lob_obj INOUT CLOB,
 source_obj IN CLOB,
 amount IN BIGINT,
 dest_offset IN BIGINT DEFAULT 1,
 src_offset IN BIGINT DEFAULT 1
)RETURN CLOB;

```

**Table 10-31** PKG\_UTIL.LOB\_COPY\_HUGE API parameters

| Parameter   | Type      | Input/Output Parameter | Can Be Empty | Description                                                                      |
|-------------|-----------|------------------------|--------------|----------------------------------------------------------------------------------|
| lob_obj     | BLOB/CLOB | INOUT                  | No           | Target BLOB/CLOB.                                                                |
| source_obj  | BLOB/CLOB | IN                     | No           | Source BLOB/CLOB.                                                                |
| amount      | BIGINT    | IN                     | No           | Length of the data to be copied (in bytes for BLOBs or in characters for CLOBs). |
| dest_offset | BIGINT    | IN                     | No           | Offset position of the target LOB to which the data is loaded.                   |
| src_offset  | BIGINT    | IN                     | No           | Offset position of the source LOB from which the data is read.                   |

- **PKG\_UTIL.BLOB\_RESET**

Sets a segment of data in a BLOB to the specified value and returns the processed BLOB and the actually processed length.

The prototype of the PKG\_UTIL.BLOB\_RESET function is as follows:

```

PKG_UTIL.BLOB_RESET(
 lob INOUT BLOB,
 len INOUT BIGINT,
 start_pos IN BIGINT DEFAULT 1,
 value IN INTEGER DEFAULT 0
)RETURN RECORD;

```

**Table 10-32** PKG\_UTIL.BLOB\_RESET API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                |
|-----------|---------|------------------------|--------------|--------------------------------------------|
| lob       | BLOB    | INOUT                  | No           | LOB to be reset.                           |
| len       | INTEGER | INOUT                  | No           | Length to reset, in bytes.                 |
| start     | INTEGER | IN                     | No           | Specifies the start position for reset.    |
| value     | INTEGER | IN                     | Yes          | Sets characters. Default value: <b>0</b> . |

- PKG\_UTIL.CLOB\_RESET  
Sets a piece of data to spaces.

The prototype of the PKG\_UTIL.CLOB\_RESET function is as follows:

```
PKG_UTIL.CLOB_RESET(
 lob INOUT CLOB,
 len INOUT BIGINT,
 start_pos IN BIGINT DEFAULT 1
)RETURN RECORD;
```

**Table 10-33** PKG\_UTIL.CLOB\_RESET API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                           |
|-----------|---------|------------------------|--------------|-------------------------------------------------------|
| lob       | CLOB    | INOUT                  | No           | LOB to be reset.                                      |
| len       | INTEGER | INOUT                  | No           | Length to reset, in characters.                       |
| start     | INTEGER | IN                     | No           | Reset start position. The default value is <b>1</b> . |

- PKG\_UTIL.LOADBLOBFROMFILE  
Reads the content of a specified length from the specified offset position of the source BFILE object, writes the content to the specified offset position of the target BLOB, and returns the target object, read position, and write position.

The prototype of the PKG\_UTIL.LOADBLOBFROMFILE function is as follows:

```
PKG_UTIL.LOADBLOBFROMFILE(
 dest_lob INOUT BLOB,
 fd IN INTEGER,
 amount IN BIGINT,
 dest_offset INOUT BIGINT,
 file_offset INOUT BIGINT
)RETURN RECORD;
```

**Table 10-34** PKG\_UTIL.LOADBLOBFROMFILE API parameters

| Parameter   | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                  |
|-------------|---------|------------------------|--------------|----------------------------------------------------------------------------------------------|
| dest_lob    | BLOB    | INOUT                  | No           | Target BLOB as the INOUT parameter.                                                          |
| fd          | INTEGER | IN                     | No           | File descriptor of the source BFILE object.                                                  |
| amount      | BIGINT  | IN                     | No           | Length of the data to be copied (in bytes for BLOBs or in characters for CLOBs).             |
| dest_offset | BIGINT  | INOUT                  | No           | Offset position of the target LOB to which the data is written as the INOUT parameter.       |
| src_offset  | BIGINT  | INOUT                  | No           | Offset position of the source BFILE file from which the data is read as the INOUT parameter. |

- **PKG\_UTIL.LOADCLOBFROMFILE**

Reads the content of a specified length from the specified offset position of the source BFILE object, writes the content to the specified offset position of the target CLOB, and returns the target object, read position, and write position.

The prototype of the PKG\_UTIL.LOADCLOBFROMFILE function is as follows:

```
PKG_UTIL.LOADCLOBFROMFILE(
 dest_lob INOUT CLOB,
 fd IN INTEGER,
 amount IN BIGINT,
 dest_offset INOUT BIGINT,
 file_offset INOUT BIGINT
)RETURN RECORD;
```

**Table 10-35** PKG\_UTIL.LOADCLOBFROMFILE API parameters

| Parameter   | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                  |
|-------------|---------|------------------------|--------------|----------------------------------------------------------------------------------------------|
| dest_lob    | CLOB    | INOUT                  | No           | Target CLOB as the INOUT parameter.                                                          |
| fd          | INTEGER | IN                     | No           | File descriptor of the source BFILE object.                                                  |
| amount      | BIGINT  | IN                     | No           | Length to copy (in characters for CLOBs).                                                    |
| dest_offset | BIGINT  | INOUT                  | No           | Offset position of the target LOB to which the data is written as the INOUT parameter.       |
| src_offset  | BIGINT  | INOUT                  | No           | Offset position of the source BFILE file from which the data is read as the INOUT parameter. |

- PKG\_UTIL.LOB\_CONVERTTOBLOB\_HUGE

Reads the content of a specified length from the specified offset position of the source CLOB, converts the content into a BLOB, and writes the BLOB to the specified position of target LOB. *amount* indicates the length to be converted.

The prototype of the PKG\_UTIL.LOB\_CONVERTTOBLOB\_HUGE function is as follows:

```
PKG_UTIL.LOB_CONVERTTOBLOB_HUGE(
 dest_lob INOUT BLOB,
 src_clob IN CLOB,
 amount IN BIGINT,
 dest_offset INOUT BIGINT,
 src_offset INOUT BIGINT)
)RETURN RECORD;
```

**Table 10-36** PKG\_UTIL.LOB\_CONVERTTOBLOB\_HUGE API parameters

| Parameter | Type | Input / Output Parameter | Can Be Empty | Description |
|-----------|------|--------------------------|--------------|-------------|
| dest_lob  | BLOB | INOUT                    | No           | Target LOB. |

| Parameter   | Type   | Input / Output Parameter | Can Be Empty | Description                                                                            |
|-------------|--------|--------------------------|--------------|----------------------------------------------------------------------------------------|
| src_clob    | CLOB   | IN                       | No           | CLOB to be converted.                                                                  |
| amount      | BIGINT | IN                       | No           | Length to convert, in characters.                                                      |
| dest_offset | BIGINT | INOUT                    | No           | Offset position of the target LOB to which the data is written as the INOUT parameter. |
| src_offset  | BIGINT | INOUT                    | No           | Offset position of the source CLOB from which the data is read as the INOUT parameter. |

- **PKG\_UTIL.LOB\_CONVERTTOCLOB\_HUGE**

Reads the content of a specified length from the specified offset position of the source CLOB, converts the content into a CLOB, and writes the CLOB to the specified position of target LOB. *amount* indicates the length to be converted.

The prototype of the PKG\_UTIL.LOB\_CONVERTTOCLOB\_HUGE function is as follows:

```
PKG_UTIL.LOB_CONVERTTOCLOB_HUGE(
 dest_lob INOUT CLOB,
 src_blob IN BLOB,
 amount IN BIGINT,
 dest_offset INOUT BIGINT,
 src_offset INOUT BIGINT)
)RETURN RECORD;
```

**Table 10-37** PKG\_UTIL.LOB\_CONVERTTOCLOB\_HUGE API parameters

| Parameter | Type | Input/ Output Parameter | Can Be Empty | Description |
|-----------|------|-------------------------|--------------|-------------|
| dest_lob  | CLOB | INOUT                   | No           | Target LOB. |

| Parameter   | Type   | Input/Output Parameter | Can Be Empty | Description                                                                            |
|-------------|--------|------------------------|--------------|----------------------------------------------------------------------------------------|
| src_blob    | BLOB   | IN                     | No           | BLOB to be converted.                                                                  |
| amount      | BIGINT | IN                     | No           | Length to convert, in bytes.                                                           |
| dest_offset | BIGINT | INOUT                  | No           | Offset position of the target LOB to which the data is written as the INOUT parameter. |
| src_offset  | BIGINT | INOUT                  | No           | Offset position of the source CLOB from which the data is read as the INOUT parameter. |

- **PKG\_UTIL.LOB\_RAWTOTEXT**

Converts RAW data to TEXT data.

The prototype of the PKG\_UTIL.LOB\_RAWTOTEXT function is as follows:

```
PKG_UTIL.LOB_RAWTOTEXT(
src_lob IN BLOB
)
RETURN TEXT
```

**Table 10-38** PKG\_UTIL.LOB\_RAWTOTEXT API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description          |
|-----------|------|------------------------|--------------|----------------------|
| src_lob   | BLOB | IN                     | No           | LOB to be converted. |

- **PKG\_UTIL.BFILE\_GET\_LENGTH**

Obtains and returns the specified length of a BFILE file.

The prototype of the PKG\_UTIL.BFILE\_GET\_LENGTH function is as follows:

```
PKG_UTIL.BFILE_GET_LENGTH(
 fd INTEGER
)RETURN BIGINT;
```

**Table 10-39** PKG\_UTIL.LOB\_GET\_LENGTH API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                  |
|-----------|---------|------------------------|--------------|----------------------------------------------|
| fd        | INTEGER | IN                     | No           | File descriptor of the specified BFILE file. |

- PKG\_UTIL.BFILE\_OPEN

Opens a BFILE file and returns its file descriptor.

The prototype of the PKG\_UTIL.BFILE\_OPEN function is as follows:

```
PKG_UTIL.BFILE_OPEN(
 file_name TEXT,
 open_mode TEXT)
RETURN INTEGER;
```

**Table 10-40** PKG\_UTIL.BFILE\_OPEN API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                                    |
|-----------|------|------------------------|--------------|----------------------------------------------------------------|
| file_name | TEXT | IN                     | No           | Name of the specified BFILE file.                              |
| open_mode | TEXT | IN                     | No           | Open mode, which can only be set to r, that is, the read mode. |

- PKG\_UTIL.BFILE\_CLOSE

Closes a BFILE file.

The prototype of the PKG\_UTIL.BFILE\_CLOSE function is as follows:

```
PKG_UTIL.BFILE_CLOSE(
 fd INTEGER)
RETURN BOOL;
```

**Table 10-41** PKG\_UTIL.BFILE\_CLOSE API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                  |
|-----------|---------|------------------------|--------------|----------------------------------------------|
| fd        | INTEGER | IN                     | No           | File descriptor of the specified BFILE file. |

- PKG\_UTIL.LOB\_WRITE\_HUGE

Reads the specified length of the source object from the start position, writes the content to the specified offset position of the target LOB, overrides the original content, and returns the target LOB.

The prototype of the PKG\_UTIL.LOB\_WRITE\_HUGE function is as follows:

```
PKG_UTIL.LOB_WRITE_HUGE(
 dest_lob INOUT BLOB,
 len IN INTEGER,
 start_pos IN BIGINT,
 src_lob IN RAW
)RETURN BLOB;
```

```
PKG_UTIL.LOB_WRITE_HUGE(
 dest_lob INOUT CLOB,
 len IN INTEGER,
 start_pos IN BIGINT,
 src_lob IN VARCHAR2
)RETURN CLOB;
```

**Table 10-42** PKG\_UTIL.LOB\_WRITE\_HUGE API parameters

| Parameter | Type         | Input/Output Parameter | Can Be Empty | Description                                                                       |
|-----------|--------------|------------------------|--------------|-----------------------------------------------------------------------------------|
| dest_lob  | BLOB/CLOB    | INOUT                  | No           | Target LOB as the INOUT parameter, to which the content is to be written.         |
| len       | INTEGER      | IN                     | No           | Length of the data to be written (in bytes for BLOBs or in characters for CLOBs). |
| start_pos | BIGINT       | IN                     | No           | Offset position for writing data to <i>dest_lob</i> .                             |
| src_lob   | RAW/VARCHAR2 | IN                     | No           | Source LOB.                                                                       |

- **PKG\_UTIL.IO\_PRINT**

Outputs a string.

The prototype of the PKG\_UTIL.IO\_PRINT function is as follows:

```
PKG_UTIL.IO_PRINT(
format IN TEXT,
is_one_line IN BOOLEAN
)
RETURN VOID;
```

**Table 10-43** PKG\_UTIL.IO\_PRINT API parameters

| Parameter   | Type    | Input/Output Parameter | Can Be Empty | Description                                       |
|-------------|---------|------------------------|--------------|---------------------------------------------------|
| format      | TEXT    | IN                     | No           | Indicates the character string to be output.      |
| is_one_line | BOOLEAN | IN                     | No           | Specifies whether to output the string as a line. |

- **PKG\_UTIL.RAW\_GET\_LENGTH**

Obtains the length of RAW data.

The prototype of the PKG\_UTIL.RAW\_GET\_LENGTH function is as follows:

```
PKG_UTIL.RAW_GET_LENGTH(
value IN RAW
)
RETURN INTEGER;
```

**Table 10-44** PKG\_UTIL.RAW\_GET\_LENGTH API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                          |
|-----------|------|------------------------|--------------|------------------------------------------------------|
| raw       | RAW  | IN                     | No           | Indicates the object whose length is to be obtained. |

- **PKG\_UTIL.RAW\_CAST\_FROM\_VARCHAR2**

Converts VARCHAR2 data to RAW data.

The prototype of the PKG\_UTIL.RAW\_CAST\_FROM\_VARCHAR2 function is as follows:

```
PKG_UTIL.RAW_CAST_FROM_VARCHAR2(
str IN VARCHAR2
)
RETURN RAW;
```

**Table 10-45** PKG\_UTIL.RAW\_CAST\_FROM\_VARCHAR2 API parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                                |
|-----------|----------|------------------------|--------------|--------------------------------------------|
| str       | VARCHAR2 | IN                     | No           | Specifies the source data to be converted. |

- PKG\_UTIL.RAW\_CAST\_FROM\_BINARY\_INTEGER

Converts BIGINT data to RAW data.

The prototype of the PKG\_UTIL.RAW\_CAST\_FROM\_BINARY\_INTEGER function is as follows:

```
PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER(
value IN BIGINT,
endianess IN INTEGER
)
RETURN RAW;
```

**Table 10-46** PKG\_UTIL.RAW\_CAST\_FROM\_BINARY\_INTEGER API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                |
|-----------|---------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| value     | BIGINT  | IN                     | No           | Specifies the source data to be converted.                                                                                                                 |
| endianess | INTEGER | IN                     | No           | The value is an integer in lexicographic order. Currently, the value can be <b>1</b> (BIG_ENDIAN), <b>2</b> (LITTLE_ENDIAN), or <b>3</b> (MACHINE_ENDIAN). |

- PKG\_UTIL.RAW\_CAST\_TO\_BINARY\_INTEGER

Converts RAW data into BINARY\_INTEGER.

The prototype of the PKG\_UTIL.RAW\_CAST\_TO\_BINARY\_INTEGER function is as follows:

```
PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER(
value IN RAW,
endianess IN INTEGER
)
RETURN INTEGER;
```

**Table 10-47** PKG\_UTIL.RAW\_CAST\_TO\_BINARY\_INTEGER API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                |
|-----------|---------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| value     | RAW     | IN                     | No           | Specifies the source data to be converted.                                                                                                                 |
| endianess | INTEGER | IN                     | No           | The value is an integer in lexicographic order. Currently, the value can be <b>1</b> (BIG_ENDIAN), <b>2</b> (LITTLE_ENDIAN), or <b>3</b> (MACHINE_ENDIAN). |

- PKG\_UTIL.RANDOM\_SET\_SEED

Sets a random seed.

The prototype of the PKG\_UTIL.RANDOM\_SET\_SEED function is as follows:

```
PKG_UTIL.RANDOM_SET_SEED(
seed IN INT
)
RETURN INTEGER;
```

**Table 10-48** PKG\_UTIL.RANDOM\_SET\_SEED API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description         |
|-----------|------|------------------------|--------------|---------------------|
| seed      | INT  | IN                     | No           | Sets a random seed. |

- PKG\_UTIL.RANDOM\_GET\_VALUE

Returns a 15-digit random number ranging from 0 to 1.

The prototype of the PKG\_UTIL.RANDOM\_GET\_VALUE function is as follows:

```
PKG_UTIL.RANDOM_GET_VALUE(
)
RETURN NUMERIC;
```

- PKG\_UTIL.FILE\_SET\_DIRNAME

Sets the directory to be operated. It must be called to set directory for each operation involving a single directory.

The prototype of the PKG\_UTIL.FILE\_SET\_DIRNAME function is as follows:

```
PKG_UTIL.FILE_SET_DIRNAME(
dir IN TEXT
)
RETURN BOOL
```

**Table 10-49** PKG\_UTIL.FILE\_SET\_DIRNAME API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                           |
|-----------|------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dirname   | TEXT | IN                     | No           | Directory of a file. It is a string, indicating an object name.<br><br><b>NOTE</b><br>File directories need to be added to the system catalog <b>PG_DIRECTORY</b> . If the input path does not match the path in <b>PG_DIRECTORY</b> , an error indicating that the path does not exist is reported. Functions that involve <b>location</b> as parameters also comply with this rule. |

- PKG\_UTIL.FILE\_OPEN

Opens a file. A maximum of 50 files can be opened at a time. This function returns a handle of the INTEGER type.

The prototype of the PKG\_UTIL.FILE\_OPEN function is as follows:

```
PKG_UTIL.FILE_OPEN(
file_name IN TEXT,
open_mode IN INTEGER)
```

**Table 10-50** PKG\_UTIL.FILE\_OPEN API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                   |
|-----------|---------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file_name | TEXT    | IN                     | No           | File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the <b>OPEN</b> function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).                   |
| open_mode | INTEGER | IN                     | No           | File opening mode, including <b>r</b> (read), <b>w</b> (write), and <b>a</b> (append).<br><br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |

- PKG\_UTIL.FILE\_SET\_MAX\_LINE\_SIZE

Sets the maximum length of a line to be written to a file.

The prototype of the PKG\_UTIL.FILE\_SET\_MAX\_LINE\_SIZE function is as follows:

```
PKG_UTIL.FILE_SET_MAX_LINE_SIZE(
max_line_size IN INTEGER)
RETURN BOOL;
```

**Table 10-51** PKG\_UTIL.FILE\_SET\_MAX\_LINE\_SIZE API parameters

| Parameter     | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                         |
|---------------|---------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| max_line_size | INTEGER | IN                     | Yes          | Maximum number of characters in each line, including newline characters. The minimum value is <b>1</b> and the maximum is <b>32767</b> . If this parameter is not specified, the default value <b>1024</b> is used. |

- PKG\_UTIL.FILE\_IS\_CLOSE

Checks whether a file handle is closed.

The prototype of the PKG\_UTIL.FILE\_IS\_CLOSE function is as follows:

```
PKG_UTIL.FILE_IS_CLOSE(
file IN INTEGER
)
RETURN BOOL;
```

**Table 10-52** PKG\_UTIL.FILE\_IS\_CLOSE API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description         |
|-----------|---------|------------------------|--------------|---------------------|
| file      | INTEGER | IN                     | No           | Opened file handle. |

- **PKG\_UTIL.FILE\_READ**

Reads a line of data from an open file handle based on the specified length.

The prototype of the PKG\_UTIL.FILE\_READ function is as follows:

```
PKG_UTIL.FILE_READ(
file IN INTEGER,
buffer OUT TEXT,
len IN BIGINT DEFAULT 1024)
```

**Table 10-53** PKG\_UTIL.FILE\_READ API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                  |
|-----------|---------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER | IN                     | No           | File handle opened by calling the OPEN function. The file must be opened in read mode. Otherwise, the INVALID_OPERATION exception is thrown. |
| buffer    | TEXT    | OUTPUT                 | No           | Buffer used to receive data.                                                                                                                 |
| len       | BIGINT  | IN                     | No           | Number of bytes read from a file.                                                                                                            |

- **PKG\_UTIL.FILE\_READLINE**

Reads a line of data from an open file handle based on the specified length.

The prototype of the PKG\_UTIL.FILE\_READLINE function is as follows:

```
PKG_UTIL.FILE_READLINE(
file IN INTEGER,
buffer OUT TEXT,
len IN INTEGER DEFAULT NULL)
```

**Table 10-54** PKG\_UTIL.FILE\_READLINE API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                           |
|-----------|---------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER | IN                     | No           | File handle opened by calling the OPEN function. The file must be opened in read mode. Otherwise, the INVALID_OPERATION exception is thrown.                          |
| buffer    | TEXT    | OUTPUT                 | No           | Buffer used to receive data.                                                                                                                                          |
| len       | INTEGER | IN                     | Yes          | Number of bytes read from a file. The default value is <b>NULL</b> . If the default value <b>NULL</b> is used, <b>max_line_size</b> is used to specify the line size. |

- PKG\_UTIL.FILE\_WRITE

Writes the data specified in the buffer to a file.

The prototype of the PKG\_UTIL.FILE\_WRITE function is as follows:

```
PKG_UTIL.FILE_WRITE(
file IN INTEGER,
buffer IN TEXT
)
RETURN BOOL;
```

**Table 10-55** PKG\_UTIL.FILE\_WRITE API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------|---------|------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER | IN                     | No           | Opened file handle.                                                                                                                                                                                                                                                                                                                                                                                   |
| buffer    | TEXT    | IN                     | No           | Text data to be written to a file. The maximum buffer size is 32767 bytes. If no value is specified, the default value is 1024 bytes. Before the writing is performed, the buffer occupied by <b>PUT</b> operations cannot exceed 32767 bytes.<br><br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |

- **PKG\_UTIL.FILE\_NEWLINE**

Writes a line terminator to an open file. The line terminator is related to the platform.

The prototype of the PKG\_UTIL.FILE\_NEWLINE function is as follows:

```
PKG_UTIL.FILE_NEWLINE(
file IN INTEGER
)
RETURN BOOL;
```

**Table 10-56** PKG\_UTIL.FILE\_NEWLINE API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description         |
|-----------|---------|------------------------|--------------|---------------------|
| file      | INTEGER | IN                     | No           | Opened file handle. |

- PKG\_UTIL.FILE\_WRITELINE

Writes a line to a file.

The prototype of the PKG\_UTIL.FILE\_WRITELINE function is as follows:

```
PKG_UTIL.FILE_WRITELINE(
file IN INTEGER,
buffer IN TEXT
)
RETURN BOOL;
```

**Table 10-57** PKG\_UTIL.FILE\_WRITELINE API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description            |
|-----------|---------|------------------------|--------------|------------------------|
| file      | INTEGER | IN                     | No           | Opened file handle.    |
| buffer    | TEXT    | IN                     | No           | Content to be written. |

- PKG\_UTIL.FILE\_READ\_RAW

Reads binary data of a specified length from an open file handle and returns the read binary data. The return type is RAW.

The prototype of the PKG\_UTIL.FILE\_READ\_RAW function is as follows:

```
PKG_UTIL.FILE_READ_RAW(
file IN INTEGER,
length IN INTEGER DEFAULT NULL
)
RETURN RAW;
```

**Table 10-58** PKG\_UTIL.FILE\_READ\_RAW API parameters

| Parameter | Type        | Input/Output Parameter | Can Be Empty | Description                                                                                                                           |
|-----------|-------------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTE<br>GER | IN                     | No           | Opened file handle.                                                                                                                   |
| length    | INTE<br>GER | IN                     | Yes          | Length of the data to be read. The default value is <b>NULL</b> . By default, all data in the file is read. The maximum size is 1 GB. |

- **PKG\_UTIL.FILE\_WRITE\_RAW**  
Writes the input binary object of the RAW type to an open file. If the insertion is successful, **true** is returned.

The prototype of the PKG\_UTIL.FILE\_WRITE\_RAW function is as follows:

```
PKG_UTIL.FILE_WRITE_RAW(
file IN INTEGER,
r IN RAW
)
RETURN BOOL;
```

**Table 10-59** PKG\_UTIL.FILE\_WRITE\_RAW API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                        |
|-----------|---------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER | IN                     | No           | Opened file handle.                                                                                                                                                                |
| r         | RAW     | IN                     | No           | Data to be written to the file.<br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |

- PKG\_UTIL.FILE\_FLUSH

Data in a file handle must be written into a physical file. Data in the buffer must have a line terminator. Refresh is important if a file must be read when it is opened. For example, debugging information can be refreshed to a file so that it can be read immediately.

The prototype of the PKG\_UTIL.FILE\_FLUSH function is as follows:

```
PKG_UTIL.FILE_FLUSH (
file IN INTEGER
)
RETURN VOID;
```

**Table 10-60** PKG\_UTIL.FILE\_FLUSH API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description         |
|-----------|---------|------------------------|--------------|---------------------|
| file      | INTEGER | IN                     | No           | Opened file handle. |

- PKG\_UTIL.FILE\_CLOSE

Closes an open file handle.

The prototype of the PKG\_UTIL.FILE\_CLOSE function is as follows:

```
PKG_UTIL.FILE_CLOSE (
file IN INTEGER
)
RETURN BOOL;
```

**Table 10-61** PKG\_UTIL.FILE\_CLOSE API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description         |
|-----------|---------|------------------------|--------------|---------------------|
| file      | INTEGER | IN                     | No           | Opened file handle. |

- PKG\_UTIL.FILE\_REMOVE

Deletes a disk file. To perform this operation, you must have required permissions.

The prototype of the PKG\_UTIL.FILE\_REMOVE function is as follows:

```
PKG_UTIL.FILE_REMOVE(
file_name IN TEXT
)
RETURN VOID;
```

**Table 10-62** PKG\_UTIL.FILE\_REMOVE API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                    |
|-----------|------|------------------------|--------------|--------------------------------|
| file_name | TEXT | IN                     | No           | Name of the file to be deleted |

- PKG\_UTIL.FILE\_RENAME

Renames a file on the disk, similar to mv in Unix.

The prototype of the PKG\_UTIL.FILE\_RENAME function is as follows:

```
PKG_UTIL.FILE_RENAME(
src_dir IN TEXT,
src_file_name IN TEXT,
dest_dir IN TEXT,
dest_file_name IN TEXT,
overwrite BOOLEAN DEFAULT FALSE)
```

**Table 10-63** PKG\_UTIL.FILE\_RENAME API parameters

| Parameter     | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|------|------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| src_dir       | TEXT | IN                     | No           | Source file directory (case-sensitive)<br><b>NOTE</b> <ul style="list-style-type: none"> <li>• Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>• When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to operate files in the file path specified by <b>safe_data_path</b>.</li> </ul> |
| src_file_name | TEXT | IN                     | No           | Source file name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

| Parameter      | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------|---------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_dir       | TEXT    | IN                     | No           | Target file directory (case-sensitive)<br><b>NOTE</b> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to operate files in the file path specified by <b>safe_data_path</b>.</li> </ul> |
| dest_file_name | TEXT    | IN                     | No           | Target file name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| overwrite      | BOOLEAN | IN                     | Yes          | The default value is <b>false</b> . If a file with the same name exists in the destination directory, the file will not be rewritten.                                                                                                                                                                                                                                                                                                                                                                               |

- **PKG\_UTIL.FILE\_SIZE**

Returns the size of a specified file.

The prototype of the **PKG\_UTIL.FILE\_SIZE** function is as follows:

```
bigint PKG_UTIL.FILE_SIZE(
file_name IN TEXT
)RETURN BIGINT;
```

**Table 10-64** PKG\_UTIL.FILE\_SIZE API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description |
|-----------|------|------------------------|--------------|-------------|
| file_name | TEXT | IN                     | No           | File name   |

- PKG\_UTIL.FILE\_BLOCK\_SIZE

Returns the number of blocks contained in a specified file.

The prototype of the PKG\_UTIL.FILE\_BLOCK\_SIZE function is as follows:

```
bigint PKG_UTIL.FILE_BLOCK_SIZE(
file_name IN TEXT
)RETURN BIGINT;
```

**Table 10-65** PKG\_UTIL.FILE\_BLOCK\_SIZE API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description |
|-----------|------|------------------------|--------------|-------------|
| file_name | TEXT | IN                     | No           | File name   |

- PKG\_UTIL.FILE\_EXISTS

Checks whether a file exists.

The prototype of the PKG\_UTIL.FILE\_EXISTS function is as follows:

```
PKG_UTIL.FILE_EXISTS(
file_name IN TEXT
)
RETURN BOOL;
```

**Table 10-66** PKG\_UTIL.FILE\_EXISTS API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description |
|-----------|------|------------------------|--------------|-------------|
| file_name | TEXT | IN                     | No           | File name   |

- PKG\_UTIL.FILE\_GETPOS

Specifies the offset of a returned file, in bytes.

The prototype of the PKG\_UTIL.FILE\_GETPOS function is as follows:

```
PKG_UTIL.FILE_GETPOS(
file IN INTEGER
)
RETURN BIGINT;
```

**Table 10-67** PKG\_UTIL.FILE\_GETPOS API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description         |
|-----------|---------|------------------------|--------------|---------------------|
| file      | INTEGER | IN                     | No           | Opened file handle. |

- PKG\_UTIL.FILE\_SEEK

Adjusts the position of a file pointer forward or backward based on the specified number of bytes.

The prototype of the PKG\_UTIL.FILE\_SEEK function is as follows:

```
void PKG_UTIL.FILE_SEEK(
file IN INTEGER,
start IN BIGINT
)
RETURN VOID;
```

**Table 10-68** PKG\_UTIL.FILE\_SEEK API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description            |
|-----------|---------|------------------------|--------------|------------------------|
| file      | INTEGER | IN                     | No           | Opened file handle.    |
| start     | BIGINT  | IN                     | No           | File offset, in bytes. |

- PKG\_UTIL.FILE\_CLOSE\_ALL

Closes all file handles opened in a session.

The prototype of the PKG\_UTIL.FILE\_CLOSE\_ALL function is as follows:

```
PKG_UTIL.FILE_CLOSE_ALL(
)
RETURN VOID;
```

**Table 10-69** PKG\_UTIL.FILE\_CLOSE\_ALL API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description |
|-----------|------|------------------------|--------------|-------------|
| None      | None | None                   | None         | None        |

- PKG\_UTIL.EXCEPTION\_REPORT\_ERROR

Throws an exception.

The prototype of the PKG\_UTIL.EXCEPTION\_REPORT\_ERROR function is as follows:

```
PKG_UTIL.EXCEPTION_REPORT_ERROR(
code INTEGER,
log TEXT,
flag BOOLEAN DEFAULT FALSE
)
RETURN INTEGER;
```

**Table 10-70** PKG\_UTIL.EXCEPTION\_REPORT\_ERROR API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                    |
|-----------|---------|------------------------|--------------|------------------------------------------------|
| code      | Integer | IN                     | No           | Error code displayed when an exception occurs. |
| log       | TEXT    | IN                     | No           | Logs displayed when an exception occurs.       |
| flag      | BOOLEAN | IN                     | Yes          | Reserved. The default value is <b>false</b> .  |

- PKG\_UTIL.APP\_READ\_CLIENT\_INFO

Reads the client information.

The prototype of the PKG\_UTIL.APP\_READ\_CLIENT\_INFO function is as follows:

```
PKG_UTIL.APP_READ_CLIENT_INFO(
OUT buffer TEXT
)RETURN TEXT;
```

**Table 10-71** PKG\_UTIL.APP\_READ\_CLIENT\_INFO API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                  |
|-----------|------|------------------------|--------------|------------------------------|
| buffer    | TEXT | OUTPUT                 | No           | Client information returned. |

- PKG\_UTIL.APP\_SET\_CLIENT\_INFO

Sets the client information.

The prototype of the PKG\_UTIL.APP\_SET\_CLIENT\_INFO function is as follows:

```
PKG_UTIL.APP_SET_CLIENT_INFO(
str TEXT
)
```

**Table 10-72** PKG\_UTIL.APP\_SET\_CLIENT\_INFO API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                   |
|-----------|------|------------------------|--------------|-------------------------------|
| str       | TEXT | IN                     | No           | Client information to be set. |

- PKG\_UTIL.LOB\_CONVERTTOBLOB

Converts a CLOB to a BLOB. *amount* indicates the conversion length.

The prototype of the PKG\_UTIL.LOB\_CONVERTTOBLOB function is as follows:

```
PKG_UTIL.LOB_CONVERTTOBLOB(
dest_lob BLOB,
src_clob CLOB,
amount INTEGER,
dest_offset INTEGER,
src_offset INTEGER
)RETURN RAW;
```

**Table 10-73** PKG\_UTIL.LOB\_CONVERTTOBLOB API parameters

| Parameter   | Type    | Input/Output Parameter | Can Be Empty | Description                        |
|-------------|---------|------------------------|--------------|------------------------------------|
| dest_lob    | BLOB    | IN                     | No           | Target LOB.                        |
| src_clob    | CLOB    | IN                     | No           | CLOB to be converted.              |
| amount      | INTEGER | IN                     | No           | Conversion length.                 |
| dest_offset | INTEGER | IN                     | No           | Start position of the target LOB.  |
| src_offset  | INTEGER | IN                     | No           | Start position of the source CLOB. |

- **PKG\_UTIL.ILOB\_CONVERTTOCLOB**

Converts a BLOB to a CLOB. *amount* indicates the conversion length.

The prototype of the PKG\_UTIL.ILOB\_CONVERTTOCLOB function is as follows:

```
PKG_UTIL.ILOB_CONVERTTOCLOB(
dest_lob CLOB,
src_blob BLOB,
amount INTEGER,
dest_offset INTEGER,
src_offset INTEGER
)RETURN TEXT;
```

**Table 10-74** PKG\_UTIL.lob\_converttoclob API parameters

| Parameter   | Type    | Input/Output Parameter | Can Be Empty | Description                        |
|-------------|---------|------------------------|--------------|------------------------------------|
| dest_lob    | CLOB    | IN                     | No           | Target LOB.                        |
| src_blob    | BLOB    | IN                     | No           | BLOB to be converted.              |
| amount      | INTEGER | IN                     | No           | Conversion length.                 |
| dest_offset | INTEGER | IN                     | No           | Start position of the target LOB.  |
| src_offset  | INTEGER | IN                     | No           | Start position of the source CLOB. |

- **PKG\_UTIL.LOB\_TEXTTORAW**

Converts the text type to the raw type.

The prototype of the PKG\_UTIL.LOB\_TEXTTORAW function is as follows:

```
PKG_UTIL.LOB_TEXTTORAW(
src_lob CLOB
)
RETURN RAW;
```

**Table 10-75** PKG\_UTIL.LOB\_TEXTTORAW API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description          |
|-----------|------|------------------------|--------------|----------------------|
| src_lob   | LOB  | IN                     | No           | LOB to be converted. |

- PKG\_UTIL.MATCH\_EDIT\_DISTANCE\_SIMILARITY**  
 Calculates the difference between two character strings.  
 The prototype of the PKG\_UTIL.MATCH\_EDIT\_DISTANCE\_SIMILARITY function is as follows:

```

PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY(
str1 TEXT,
str2 TEXT
)
RETURN INTEGER;

```

**Table 10-76** PKG\_UTIL.MATCH\_EDIT\_DISTANCE\_SIMILARITY API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description              |
|-----------|------|------------------------|--------------|--------------------------|
| str1      | TEXT | IN                     | No           | First character string.  |
| str2      | TEXT | IN                     | No           | Second character string. |

- PKG\_UTIL.RAW\_CAST\_TO\_VARCHAR2**  
 Converts the raw type to the varchar2 type.  
 The prototype of the PKG\_UTIL.RAW\_CAST\_TO\_VARCHAR2 function is as follows:

```
PKG_UTIL.RAW_CAST_TO_VARCHAR2(
str RAW
)
RETURN VARCHAR2;
```

**Table 10-77** PKG\_UTIL.RAW\_CAST\_TO\_VARCHAR2 API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description         |
|-----------|------|------------------------|--------------|---------------------|
| str       | RAW  | IN                     | No           | Hexadecimal string. |

- PKG\_UTIL.SESSION\_CLEAR\_CONTEXT

Clears the session context.

The prototype of the PKG\_UTIL.SESSION\_CLEAR\_CONTEXT function is as follows:

```
PKG_UTIL.SESSION_CLEAR_CONTEXT(
namespace TEXT,
client_identifier TEXT,
attribute TEXT
)
RETURN INTEGER;
```

**Table 10-78** PKG\_UTIL.SESSION\_CLEAR\_CONTEXT API parameters

| Parameter         | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                               |
|-------------------|------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| namespace         | TEXT | IN                     | No           | Namespace of an attribute.                                                                                                                                                |
| client_identifier | TEXT | IN                     | Yes          | Usually the value of <b>client_identifier</b> is the same as that of <b>namespace</b> . If this parameter is set to <b>null</b> , all namespaces are modified by default. |

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                    |
|-----------|------|------------------------|--------------|--------------------------------|
| attribute | TEXT | IN                     | No           | Attribute value to be cleared. |

- **PKG\_UTIL.SESSION\_SEARCH\_CONTEXT**

Searches for an attribute value.

The prototype of the PKG\_UTIL.SESSION\_SEARCH\_CONTEXT function is as follows:

```
PKG_UTIL.SESSION_SEARCH_CONTEXT(
namespace TEXT,
attribute TEXT
)
RETURN INTEGER;
```

**Table 10-79** PKG\_UTIL.SESSION\_SEARCH\_CONTEXT API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                         |
|-----------|------|------------------------|--------------|-------------------------------------|
| namespace | TEXT | IN                     | No           | Namespace of an attribute.          |
| attribute | TEXT | IN                     | No           | Attribute value to be searched for. |

- **PKG\_UTIL.SESSION\_SET\_CONTEXT**

Sets an attribute value.

The prototype of the PKG\_UTIL.SESSION\_SET\_CONTEXT function is as follows:

```
PKG_UTIL.SESSION_SET_CONTEXT(
namespace TEXT,
attribute TEXT,
value TEXT
```

```
)
RETURN INTEGER;
```

**Table 10-80** PKG\_UTIL.SESSION\_SET\_CONTEXT API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                |
|-----------|------|------------------------|--------------|----------------------------|
| namespace | TEXT | IN                     | No           | Namespace of an attribute. |
| attribute | TEXT | IN                     | No           | Attribute to be set.       |
| value     | TEXT | IN                     | No           | Attribute value.           |

- **PKG\_UTIL.UTILITY\_COMPILE\_SCHEMA**

Recompiles packages, functions and stored procedures under the specified schema.

The prototype of the PKG\_UTIL.UTILITY\_COMPILE\_SCHEMA function is as follows:

```
PKG_UTIL.UTILITY_COMPILE_SCHEMA (
schema IN VARCHAR2,
compile_all IN BOOLEAN DEFAULT TRUE,
reuse_settings IN BOOLEAN DEFAULT FALSE
)
RETURNS VOID
```

- **PKG\_UTIL.GS\_COMPILE\_SCHEMA**

Recompiles packages, functions and stored procedures under the specified schema.

The prototype of the PKG\_UTIL.GS\_COMPILE\_SCHEMA stored procedure is as follows:

```
pkg_util.GS_COMPILE_SCHEMA (
schema_name IN VARCHAR2 DEFAULT NULL,
compile_all IN BOOLEAN DEFAULT FALSE,
retry_times IN INT DEFAULT 10
)
```

**Table 10-81** PKG\_UTIL.GS\_COMPILE\_SCHEMA API parameters

| Parameter   | Type     | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                           |
|-------------|----------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| schema_name | VARCHAR2 | IN                     | Yes          | Specifies the name of the namespace.                                                                                                                                                                                                                                                                                                                                  |
| compile_all | Boolean  | IN                     | Yes          | Specifies to compile all objects. <ul style="list-style-type: none"> <li>• <b>false</b>: Compile the packages, functions, and stored procedures whose status is false in the pg_object table.</li> <li>• <b>true</b>: Compile all packages, functions, and stored procedures in the pg_object table.</li> <li>• <b>retry_times</b>: number of retry times.</li> </ul> |

- PKG\_UTIL.UTILITY\_GET\_TIME

Prints the Unix timestamp.

The prototype of the PKG\_UTIL.UTILITY\_GET\_TIME function is as follows:

```
PKG_UTIL.UTILITY_GET_TIME()
RETURN BIGINT;
```

- PKG\_UTIL.UTILITY\_FORMAT\_ERROR\_BACKTRACE

Displays the error stack of a stored procedure.

The prototype of the PKG\_UTIL.UTILITY\_FORMAT\_ERROR\_BACKTRACE function is as follows:

```
PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE()
RETURN TEXT;
```

- PKG\_UTIL.UTILITY\_FORMAT\_ERROR\_STACK

Displays the error information about a stored procedure.

The prototype of the PKG\_UTIL.UTILITY\_FORMAT\_ERROR\_STACK function is as follows:

```
PKG_UTIL.UTILITY_FORMAT_ERROR_STACK()
RETURN TEXT;
```

- PKG\_UTIL.UTILITY\_FORMAT\_CALL\_STACK

Displays the call stack of a stored procedure.

The prototype of the PKG\_UTIL.UTILITY\_FORMAT\_CALL\_STACK function is as follows:

```
PKG_UTIL.UTILITY_FORMAT_CALL_STACK()
RETURN TEXT;
```

### 10.11.1.3 DBE\_XML

**Table 10-82** lists all APIs supported by the DBE\_XML API.

**Table 10-82** DBE\_XML parameters

| API                                               | Description                                                                               |
|---------------------------------------------------|-------------------------------------------------------------------------------------------|
| <a href="#">DBE_XML.XML_FREE_PARSER</a>           | Frees a parser.                                                                           |
| <a href="#">DBE_XML.XML_PARSER_GET_DOCUMENT</a>   | Obtains the parsed document node.                                                         |
| <a href="#">DBE_XML.XML_GET_VALIDATION_MODE</a>   | Obtains the validation attribute.                                                         |
| <a href="#">DBE_XML.XML_NEW_PARSER</a>            | Creates a parser instance.                                                                |
| <a href="#">DBE_XML.XML_PARSE_BUFFER</a>          | Parses the VARCHAR string.                                                                |
| <a href="#">DBE_XML.XML_PARSE_CLOB</a>            | Parses the CLOB string.                                                                   |
| <a href="#">DBE_XML.XML_SET_VALIDATION_MODE</a>   | Sets the validation attribute.                                                            |
| <a href="#">DBE_XML.XML_DOM_APPEND_CHILD</a>      | Adds the newchild node to the end of the parent(n) node and returns the newly added node. |
| <a href="#">DBE_XML.XML_DOM_CREATE_ELEMENT</a>    | Returns the DOMELEMENT object with the specified name.                                    |
| <a href="#">DBE_XML.XML_DOM_CREATE_ELEMENT_NS</a> | Returns the DOMELEMENT object with the specified name and namespace.                      |
| <a href="#">DBE_XML.XML_DOM_CREATE_TEXT_NODE</a>  | Creates and returns a DOMText object.                                                     |
| <a href="#">DBE_XML.XML_DOM_FREE_DOCUMENT</a>     | Frees a specified XML DOM object.                                                         |
| <a href="#">DBE_XML.XML_DOM_FREE_ELEMENT</a>      | Frees a specified XML DOM object.                                                         |
| <a href="#">DBE_XML.XML_DOM_FREE_NODE</a>         | Frees a DOMNode node.                                                                     |
| <a href="#">DBE_XML.XML_DOM_FREE_NODELIST</a>     | Frees a DOMNodeList node.                                                                 |
| <a href="#">DBE_XML.XML_DOM_GET_ATTRIBUTES</a>    | Obtains the attributes of a specified XML DOM object.                                     |
| <a href="#">DBE_XML.XML_DOM_GET_ATTRIBUTES</a>    | Returns the attribute values of a DOMNode node as a map.                                  |
| <a href="#">DBE_XML.XML_DOM_GET_CHILD_NODES</a>   | Converts several subnodes under a node into a node list.                                  |

| API                                                        | Description                                                                            |
|------------------------------------------------------------|----------------------------------------------------------------------------------------|
| <a href="#">DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME</a>    | Obtains the list of specified subnodes of a specified XML DOM object.                  |
| <a href="#">DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS</a> | Obtains the list of subnodes in the specified namespace of a specified XML DOM object. |
| <a href="#">DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT</a>       | Returns the first subnode of the specified document.                                   |
| <a href="#">DBE_XML.XML_DOM_GET_FIRST_CHILD</a>            | Returns the first subnode of a node.                                                   |
| <a href="#">DBE_XML.XML_DOM_GET_LAST_CHILD</a>             | Returns the last subnode of a node.                                                    |
| <a href="#">DBE_XML.XML_DOM_GET_LENGTH</a>                 | Returns the number of nodes based on the content in the node of the specified type.    |
| <a href="#">DBE_XML.XML_DOM_GET_LOCALNAME</a>              | Returns the local name of the given object.                                            |
| <a href="#">DBE_XML.XML_DOM_GET_NAMED_ITEM</a>             | Returns the node specified by name.                                                    |
| <a href="#">DBE_XML.XML_DOM_GET_NAMED_ITEM_NS</a>          | Returns the node specified by name and namespace.                                      |
| <a href="#">DBE_XML.XML_DOM_GET_NEXT_SIBLING</a>           | Returns the next node of the specified node.                                           |
| <a href="#">DBE_XML.XML_DOM_GET_NODE_NAME</a>              | Returns the name of a node.                                                            |
| <a href="#">DBE_XML.XML_DOM_GET_NODE_TYPE</a>              | Returns the type of a node.                                                            |
| <a href="#">DBE_XML.XML_DOM_GET_NODE_VALUE</a>             | Returns the value of a node.                                                           |
| <a href="#">DBE_XML.XML_DOM_GET_PARENT_NODE</a>            | Returns the parent node of the given node.                                             |
| <a href="#">DBE_XML.XML_DOM_GET_TAGNAME</a>                | Obtains the tag name of a specified XML DOM object.                                    |
| <a href="#">DBE_XML.XML_DOM_HAS_CHILD_NODES</a>            | Checks whether the DOMNode object has any subnode.                                     |
| <a href="#">DBE_XML.XML_DOM_IMPORT_NODE</a>                | Copies a node to another node and mounts the copied node to a specified document.      |
| <a href="#">DBE_XML.XML_DOM_IS_NULL</a>                    | Checks whether the given object is null.                                               |
| <a href="#">DBE_XML.XML_DOM_ITEM</a>                       | Returns the element corresponding to the index in a list or map based on the index.    |

| API                                                     | Description                                                                                            |
|---------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_XML.XML_DOM_MAKE_ELEMENT</a>            | Returns the DOMELEMENT object after conversion.                                                        |
| <a href="#">DBE_XML.XML_DOM_MAKENODE</a>                | Converts the given object to the DOMNode type.                                                         |
| <a href="#">DBE_XML.XML_DOM_NEW_DOCUMENT_EMPTY</a>      | Returns a new DOMDocument object.                                                                      |
| <a href="#">DBE_XML.XML_DOM_NEW_DOCUMENT_CLOB</a>       | Returns a new DOMDocument instance object created from the specified CLOB type.                        |
| <a href="#">DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE</a>    | Returns a new DOMDocument instance object created from the specified XMLType type.                     |
| <a href="#">DBE_XML.XML_DOM_SET_ATTRIBUTE</a>           | Sets the attributes of a specified XML DOM object.                                                     |
| <a href="#">DBE_XML.XML_DOM_SET_CHARACTERSET</a>        | Sets the character set for a DOMDocument object.                                                       |
| <a href="#">DBE_XML.XML_DOM_SET_DOCTYPE</a>             | Sets the external DTD of a DOMDocument object.                                                         |
| <a href="#">DBE_XML.XML_DOM_SET_NODE_VALUE</a>          | Sets the value of a node in the DOMNode object.                                                        |
| <a href="#">DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC</a>     | Writes the given DOMDocument object to the buffer.                                                     |
| <a href="#">DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE</a>    | Writes the given DOMNode object to the buffer.                                                         |
| <a href="#">DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC</a>       | Writes the given DOMDocument object to a CLOB.                                                         |
| <a href="#">DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE</a>      | Writes the given DOMNode object to a CLOB.                                                             |
| <a href="#">DBE_XML.XML_DOM_WRITE_TO_FILE_DOC</a>       | Writes an XML node to a specified file using the database character set.                               |
| <a href="#">DBE_XML.XML_DOM_WRITE_TO_FILE_NODE</a>      | Writes an XML node to a specified file using the database character set.                               |
| <a href="#">DBE_XML.XML_DOM_GET_SESSION_TREE_NUM</a>    | Displays the number of DOM trees of all types in the current session.                                  |
| <a href="#">DBE_XML.XML_DOM_GET_DOCUMENT_TREES_INFO</a> | Displays statistics such as the memory usage and number of nodes of the DOM tree of the document type. |
| <a href="#">DBE_XML.XML_DOM_GET_DOCUMENT_TREE_INFO</a>  | Displays the number of nodes of each type for a specific document variable.                            |

- DBE\_XML.XML\_FREE\_PARSER

Frees a given parser object.

The stored procedure prototype of DBE\_XML.XML\_FREE\_PARSER is as follows:

```
DBE_XML.XML_FREE_PARSER(
id IN RAW(13))
returns VOID;
```

**Table 10-83** DBE\_XML.XML\_FREE\_PARSER parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description    |
|-----------|---------|------------------------|--------------|----------------|
| id        | RAW(13) | IN                     | Yes          | Parser object. |

- DBE\_XML.XML\_PARSER\_GET\_DOC

Returns the root node of the DOM tree document constructed by the parser.

The prototype of the DBE\_XML.XML\_PARSER\_GET\_DOC function is as follows:

```
DBE_XML.XML_PARSER_GET_DOC(
id IN RAW(13))
returns RAW(13);
```

**Table 10-84** DBE\_XML.XML\_PARSER\_GET\_DOC parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description    |
|-----------|---------|------------------------|--------------|----------------|
| id        | RAW(13) | IN                     | Yes          | Parser object. |

 **NOTE**

- If the DBE\_XML.XML\_PARSER\_GET\_DOC function is empty, null is returned.
- If the parser input by the DBE\_XML.XML\_PARSER\_GET\_DOC function has not parsed any document, null is returned.
- DBE\_XML.XML\_GET\_VALIDATION\_MODE

Obtains the parsing validation mode of a specified parser. If DTD validation is enabled, **TRUE** is returned. Otherwise, **FALSE** is returned.

The prototype of the DBE\_XML.XML\_GET\_VALIDATION\_MODE function is as follows:

```
DBE_XML.XML_GET_VALIDATION_MODE(
id RAW(13))
RETURNS BOOL;
```

**Table 10-85** DBE\_XML.XML\_GET\_VALIDATION\_MODE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description    |
|-----------|---------|------------------------|--------------|----------------|
| id        | RAW(13) | IN                     | Yes          | Parser object. |

- DBE\_XML.XML\_NEW\_PARSER

Creates a parser object and returns a new parser instance.

The prototype of the DBE\_XML.XML\_NEW\_PARSER function is as follows:

```
DBE_XML.XML_NEW_PARSER()
RETURNS RAW(13);
```

- DBE\_XML.XML\_PARSE\_BUFFER

Parses XML documents stored in strings.

The stored procedure prototype of DBE\_XML.XML\_PARSE\_BUFFER is as follows:

```
DBE_XML.XML_PARSE_BUFFER(
id RAW(13),
xmlstr VARCHAR2)
RETURNS VOID;
```

**Table 10-86** DBE\_XML.XML\_PARSE\_BUFFER parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                         |
|-----------|----------|------------------------|--------------|-------------------------------------|
| id        | RAW(13)  | IN                     | Yes          | Parser object                       |
| xmlstr    | VARCHAR2 | IN                     | No           | A string that stores XML documents. |

 **NOTE**

- The maximum length of a character string that can be parsed by the XML\_PARSE\_BUFFER function is 32767. If the length exceeds the maximum, an error is reported.
  - Different from the ORA database, this database supports only UTF-8 in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsed, a warning appears but the execution is normal. For versions later than 1.9, an error is reported.
  - DTD validation differences:
    - **!ATTLIST to type (CHECK|check|Check) "Ch..."** reports an error because the default value "Ch..." is not an enumerated value in the brackets. However, the ORA database does not report this error.
    - **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer** reports an error because the letters are case-sensitive. **Baidu** cannot correspond to **baidu**. However, the ORA database does not report this error.
  - Namespace validation difference: Undeclared namespace tags are parsed. However, the ORA database reports an error.
  - Difference in parsing XML predefined entities: **&apos;** and **&quot;** are parsed and translated into ' and ". However, predefined entities in database ORA are not translated into characters.
- **DBE\_XML.XML\_PARSE\_CLOB**

Parses XML documents stored in XML\_PARSE\_CLOB.

The stored procedure prototype of DBE\_XML.XML\_PARSE\_CLOB is as follows:

```
DBE_XML.XML_PARSE_CLOB(
id IN RAW(13),
doc IN CLOB)
RETURNS VOID;
```

**Table 10-87** DBE\_XML.XML\_PARSE\_CLOB parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                         |
|-----------|---------|------------------------|--------------|-------------------------------------|
| id        | RAW(13) | IN                     | Yes          | Parser object                       |
| doc       | CLOB    | IN                     | No           | A string that stores XML documents. |

 NOTE

- XML\_PARSE\_CLOB cannot parse CLOBs larger than 1 GB.
- Different from the ORA database, this database supports only UTF-8 in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsed, a warning appears but the execution is normal. For versions later than 1.9, an error is reported.
- DTD validation differences:
  - **!ATTLIST to type (CHECK|check|Check) "Ch..."** reports an error because the default value "Ch..." is not an enumerated value in the brackets. However, the ORA database does not report this error.
  - **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer** reports an error because the letters are case-sensitive. **Baidu** cannot correspond to **baidu**. However, the ORA database does not report this error.
- Namespace validation difference: Undeclared namespace tags are parsed. However, the ORA database reports an error.
- Difference in parsing XML predefined entities: **&apos;** and **&quot;** are parsed and translated into ' and ". However, predefined entities in database ORA are not translated into characters.

- DBE\_XML.XML\_SET\_VALIDATION\_MODE

Sets the parsing validation mode of a specified parser.

The stored procedure prototype of DBE\_XML.XML\_SET\_VALIDATION\_MODE is as follows:

```
DBE_XML.XML_SET_VALIDATION_MODE(
id RAW(13),
validate BOOLEAN)
RETURNS VOID;
```

**Table 10-88** DBE\_XML.XML\_SET\_VALIDATION\_MODE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                      |
|-----------|---------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id        | RAW(13) | IN                     | Yes          | Parser object                                                                                                                                                    |
| validate  | BOOLEAN | IN                     | Yes          | Mode to be set: <ul style="list-style-type: none"> <li>• <b>TRUE</b>: DTD validation is enabled.</li> <li>• <b>FALSE</b>: DTD validation is disabled.</li> </ul> |

 NOTE

- If the input parameter **validate** of the XML\_SET\_VALIDATION\_MODE function is null, the parsing validation mode of the parser is not changed.
- By default, the DTD validation is enabled during parser initialization.
- DBE\_XML.XML\_DOM\_APPEND\_CHILD  
Adds the newchild node to the end of the parent(n) node and returns the newly added node.

The stored procedure prototype of DBE\_XML.XML\_DOM\_APPEND\_CHILD is as follows:

```
DBE_XML.XML_DOM_APPEND_CHILD(
 parentId IN RAW(13),
 childId IN RAW(13)
)
RETURNS RAW(13);
```

**Table 10-89** DBE\_XML.XML\_DOM\_APPEND\_CHILD parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| parentId  | RAW(13) | IN                     | No           | XML DOM object  |
| childId   | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_CREATE\_ELEMENT

Returns the DOMELEMENT object with the specified name.

The prototype of the DBE\_XML.XML\_DOM\_CREATE\_ELEMENT function is as follows:

```
DBE_XML.XML_DOM_CREATE_ELEMENT(
 id IN RAW(13),
 tagname IN VARCHAR2
)
RETURNS RAW(13);
```

**Table 10-90** DBE\_XML.XML\_DOM\_CREATE\_ELEMENT parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                        |
|-----------|----------|------------------------|--------------|------------------------------------|
| id        | RAW(13)  | IN                     | No           | XML DOM object                     |
| tagname   | VARCHAR2 | IN                     | No           | Name of the new DOMELEMENT object. |

- DBE\_XML.XML\_DOM\_CREATE\_ELEMENT\_NS

Returns the DOMELEMENT object with the specified name and namespace.

The prototype of the DBE\_XML.XML\_DOM\_CREATE\_ELEMENT\_NS function is as follows:

```
DBE_XML.XML_DOM_CREATE_ELEMENT_NS(
 id IN RAW(13),
 tagname IN VARCHAR2,
 ns IN VARCHAR2
)
RETURNS RAW(13);
```

**Table 10-91** DBE\_XML.XML\_DOM\_CREATE\_ELEMENT\_NS parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                       |
|-----------|----------|------------------------|--------------|-----------------------------------|
| id        | RAW(13)  | IN                     | No           | XML DOM object                    |
| tagname   | VARCHAR2 | IN                     | No           | Name of the new DOMElement object |
| ns        | VARCHAR2 | IN                     | No           | Namespace.                        |

- DBE\_XML.XML\_DOM\_CREATE\_TEXT\_NODE

Creates and returns a DOMText object.

The prototype of the DBE\_XML.XML\_DOM\_CREATE\_TEXT\_NODE function is as follows:

```
DBE_XML.XML_DOM_CREATE_TEXT_NODE(
 id IN RAW(13),
 data IN VARCHAR2
)
RETURNS RAW(13);
```

**Table 10-92** DBE\_XML.XML\_DOM\_CREATE\_TEXT\_NODE parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                      |
|-----------|----------|------------------------|--------------|----------------------------------|
| id        | RAW(13)  | IN                     | No           | XML DOM object                   |
| data      | VARCHAR2 | IN                     | No           | Content of the new DOMText node. |

- DBE\_XML.XML\_DOM\_FREE\_DOCUMENT

Frees a specified XML DOM object.

The stored procedure prototype of DBE\_XML.XML\_DOM\_FREE\_DOCUMENT is as follows:

```
DBE_XML.XML_DOM_FREE_DOCUMENT(
 id RAW(13)
)
RETURNS VOID;
```

**Table 10-93** DBE\_XML.XML\_DOM\_FREE\_DOCUMENT parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_FREE\_ELEMENT

Frees a specified XML DOM object.

The stored procedure prototype of DBE\_XML.XML\_DOM\_FREE\_ELEMENT is as follows:

```
DBE_XML.XML_DOM_FREE_ELEMENT (
 id RAW(13)
)
RETURNS VOID;
```

**Table 10-94** DBE\_XML.XML\_DOM\_FREE\_ELEMENT parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_FREE\_NODE

Frees a DOMNode node.

The prototype of the DBE\_XML.XML\_DOM\_FREE\_NODE function is as follows:

```
DBE_XML.XML_DOM_FREE_NODE (
 id RAW(13)
)
RETURNS VOID;
```

**Table 10-95** DBE\_XML.XML\_DOM\_FREE\_NODE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_FREE\_NODELIST

Frees a DOMNodeList node.

The stored procedure prototype of DBE\_XML.XML\_DOM\_FREE\_NODELIST is as follows:

```
DBE_XML.XML_DOM_FREE_NODELIST(
 id IN RAW(13)
)
RETURNS VOID;
```

**Table 10-96** dbe\_xml.xml\_dom\_free\_nodelist parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- **DBE\_XML.XML\_DOM\_GET\_ATTRIBUTE**

Obtains the attributes of a specified XML DOM object.

The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_ATTRIBUTE is as follows:

```
DBE_XML.XML_DOM_GET_ATTRIBUTE (
 docid IN RAW(13),
 name IN VARCHAR2
)
RETURNS VARCHAR2;
```

**Table 10-97** DBE\_XML.XML\_DOM\_GET\_ATTRIBUTE parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description    |
|-----------|----------|------------------------|--------------|----------------|
| docid     | RAW(13)  | IN                     | No           | XML DOM object |
| name      | VARCHAR2 | IN                     | No           | String.        |

- **DBE\_XML.XML\_DOM\_GET\_ATTRIBUTES**

Returns the attribute values of a DOMNode node as a map.

The prototype of the DBE\_XML.XML\_DOM\_GET\_ATTRIBUTES function is as follows:

```
DBE_XML.XML_DOM_GET_ATTRIBUTES (
 id RAW(13)
)
RETURNS RAW(13);
```

**Table 10-98** DBE\_XML.XML\_DOM\_GET\_ATTRIBUTES parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- **DBE\_XML.XML\_DOM\_GET\_CHILD\_NODES**

Converts several subnodes under a node into a node list.

The prototype of the DBE\_XML.XML\_DOM\_GET\_CHILD\_NODES function is as follows:

```
DBE_XML.XML_DOM_GET_CHILD_NODES(
 id IN RAW(13)
)
RETURNS RAW(13);
```

**Table 10-99** DBE\_XML.XML\_DOM\_GET\_CHILD\_NODES parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_CHILDREN\_BY\_TAGNAME  
 Obtains the list of specified subnodes of a specified XML DOM object.  
 The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_CHILDREN\_BY\_TAGNAME is as follows:

```
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME (
 docid IN RAW(13),
 name IN VARCHAR2
)
RETURNS RAW(13);
```

**Table 10-100** DBE\_XML.XML\_DOM\_GET\_CHILDREN\_BY\_TAGNAME parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description    |
|-----------|----------|------------------------|--------------|----------------|
| docid     | RAW(13)  | IN                     | No           | XML DOM object |
| name      | VARCHAR2 | IN                     | No           | String.        |

- DBE\_XML.XML\_DOM\_GET\_CHILDREN\_BY\_TAGNAME\_NS  
 Obtains the list of subnodes in the specified namespace of a specified XML DOM object.  
 The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_CHILDREN\_BY\_TAGNAME\_NS is as follows:

```
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS (
 docid IN RAW(13),
 name IN VARCHAR2,
 ns IN VARCHAR2
)
RETURNS RAW(13);
```

**Table 10-101** DBE\_XML.XML\_DOM\_GET\_CHILDREN\_BY\_TAGNAME\_NS parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description    |
|-----------|----------|------------------------|--------------|----------------|
| docid     | RAW(13)  | IN                     | No           | XML DOM object |
| name      | VARCHAR2 | IN                     | No           | String         |
| ns        | VARCHAR2 | IN                     | Yes          | String.        |

- DBE\_XML.XML\_DOM\_GET\_DOCUMENT\_ELEMENT

Returns the first subnode of the specified document.

The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_DOCUMENT\_ELEMENT is as follows:

```
DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT(
 id RAW(13)
)
RETURNS RAW(13);
```

**Table 10-102** DBE\_XML.XML\_DOM\_GET\_DOCUMENT\_ELEMENT parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_FIRST\_CHILD

Returns the first subnode of a node.

The prototype of the DBE\_XML.XML\_DOM\_GET\_FIRST\_CHILD function is as follows:

```
DBE_XML.XML_DOM_GET_FIRST_CHILD(
 id IN RAW(13)
)
RETURNS RAW(13);
```

**Table 10-103** DBE\_XML.XML\_DOM\_GET\_FIRST\_CHILD parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_LAST\_CHILD

Returns the last subnode of a node.

The prototype of the DBE\_XML.XML\_DOM\_GET\_LAST\_CHILD function is as follows:

```
DBE_XML.XML_DOM_GET_LAST_CHILD(
 id IN RAW(13)
)
RETURNS RAW(13);
```

**Table 10-104** DBE\_XML.XML\_DOM\_GET\_LAST\_CHILD parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_LENGTH

Returns the number of nodes based on the content in the node of the specified type.

The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_LENGTH is as follows:

```
DBE_XML.XML_DOM_GET_LENGTH(
 id RAW(13)
)
RETURNS VOID;
```

**Table 10-105** DBE\_XML.XML\_DOM\_GET\_LENGTH parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_LOCALNAME

Returns the local name of the given object.

The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_LOCALNAME is as follows:

```
DBE_XML.XML_DOM_GET_LOCALNAME (
 id RAW(13)
)
RETURNS VARCHAR2;
```

**Table 10-106** DBE\_XML.XML\_DOM\_GET\_LOCALNAME parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_NAMED\_ITEM

Returns the node specified by name.

The prototype of the DBE\_XML.XML\_DOM\_GET\_NAMED\_ITEM function is as follows:

```
DBE_XML.XML_DOM_GET_NAMED_ITEM(
 id IN RAW(13),
 nodeName IN VARCHAR2
)
RETURNS RAW(13);
```

**Table 10-107** DBE\_XML.XML\_DOM\_GET\_NAMED\_ITEM parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                          |
|-----------|----------|------------------------|--------------|--------------------------------------|
| id        | RAW(13)  | IN                     | No           | XML DOM object                       |
| nodeName  | VARCHAR2 | IN                     | No           | Name of the element to be retrieved. |

- DBE\_XML.XML\_DOM\_GET\_NAMED\_ITEM\_NS

Returns the node specified by name and namespace.

The prototype of the DBE\_XML.XML\_DOM\_GET\_NAMED\_ITEM\_NS function is as follows:

```
DBE_XML.XML_DOM_GET_NAMED_ITEM_NS(
 id RAW(13),
 nodeName IN VARCHAR2,
 ns IN VARCHAR2
)
RETURNS RAW(13);
```

**Table 10-108** DBE\_XML.XML\_DOM\_GET\_NAMED\_ITEM\_NS parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                         |
|-----------|----------|------------------------|--------------|-------------------------------------|
| id        | RAW(13)  | IN                     | No           | XML DOM object                      |
| nodeName  | VARCHAR2 | IN                     | No           | Name of the element to be retrieved |
| ns        | VARCHAR2 | IN                     | Yes          | Namespace.                          |

- DBE\_XML.XML\_DOM\_GET\_NEXT\_SIBLING

Returns the next node of the specified node.

The prototype of the DBE\_XML.XML\_DOM\_GET\_NEXT\_SIBLING function is as follows:

```
DBE_XML.XML_DOM_GET_NEXT_SIBLING(
 id IN RAW(13)
)
RETURNS RAW(13);
```

**Table 10-109** DBE\_XML.XML\_DOM\_GET\_NEXT\_SIBLING parameters

| Parameter | Type    | Input/<br>Output<br>Parameter | Can Be<br>Empty | Description     |
|-----------|---------|-------------------------------|-----------------|-----------------|
| id        | RAW(13) | IN                            | No              | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_NODE\_NAME

Returns the name of a node.

The prototype of the DBE\_XML.XML\_DOM\_GET\_NODE\_NAME function is as follows:

```
DBE_XML.XML_DOM_GET_NODE_NAME(
 id IN RAW(13)
)
RETURNS VARCHAR2;
```

**Table 10-110** DBE\_XML.XML\_DOM\_GET\_NODE\_NAME parameters

| Parameter | Type    | Input/<br>Output<br>Parameter | Can Be<br>Empty | Description     |
|-----------|---------|-------------------------------|-----------------|-----------------|
| id        | RAW(13) | IN                            | No              | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_NODE\_TYPE

Returns the type of a node.

The prototype of the DBE\_XML.XML\_DOM\_GET\_NODE\_TYPE function is as follows:

```
DBE_XML.XML_DOM_GET_NODE_TYPE(
 id IN RAW(13)
)
RETURNS INTEGER;
```

**Table 10-111** DBE\_XML.XML\_DOM\_GET\_NODE\_TYPE parameters

| Parameter | Type    | Input/<br>Output<br>Parameter | Can Be<br>Empty | Description     |
|-----------|---------|-------------------------------|-----------------|-----------------|
| id        | RAW(13) | IN                            | No              | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_NODE\_VALUE

Returns the value of a node.

The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_NODE\_VALUE is as follows:

```
DBE_XML.XML_DOM_GET_NODE_VALUE(
id IN RAW(13))
RETURNS VARCHAR2;
```

**Table 10-112** DBE\_XML.XML\_DOM\_GET\_NODE\_VALUE parameters

| Parameter | Type    | Input/<br>Output<br>Parameter | Can Be<br>Empty | Description     |
|-----------|---------|-------------------------------|-----------------|-----------------|
| id        | RAW(13) | IN                            | Yes             | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_PARENT\_NODE

Returns the parent node of the given node.

The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_PARENT\_NODE is as follows:

```
DBE_XML.XML_DOM_GET_PARENT_NODE(
id IN RAW(13))
RETURNS RAW(13);
```

**Table 10-113** DBE\_XML.XML\_DOM\_GET\_PARENT\_NODE parameters

| Parameter | Type    | Input/<br>Output<br>Parameter | Can Be<br>Empty | Description     |
|-----------|---------|-------------------------------|-----------------|-----------------|
| id        | RAW(13) | IN                            | Yes             | XML DOM object. |

- DBE\_XML.XML\_DOM\_GET\_TAGNAME

Obtains the tag name of a specified XML DOM object.

The stored procedure prototype of DBE\_XML.XML\_DOM\_GET\_TAGNAME is as follows:

```
DBE_XML.XML_DOM_GET_TAGNAME (
docid RAW(13)
)
RETURNS VARCHAR2;
```

**Table 10-114** DBE\_XML.XML\_DOM\_GET\_TAGNAME parameters

| Parameter | Type    | Input/<br>Output<br>Parameter | Can Be<br>Empty | Description     |
|-----------|---------|-------------------------------|-----------------|-----------------|
| docid     | RAW(13) | IN                            | Yes             | XML DOM object. |

- DBE\_XML.XML\_DOM\_HAS\_CHILD\_NODES

Checks whether the DOMNode object has any subnode.

The stored procedure prototype of DBE\_XML.XML\_DOM\_HAS\_CHILD\_NODES is as follows:

```
DBE_XML.XML_DOM_HAS_CHILD_NODES(
id IN RAW(13))
RETURNS BOOLEAN;
```

**Table 10-115** DBE\_XML.XML\_DOM\_HAS\_CHILD\_NODES parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_IMPORT\_NODE

Copies a node to another node and mounts the copied node to a specified document. If the type of the copied node does not belong to the 12 types specified by constants of XML DOM, an exception indicating that the type is not supported is thrown.

The prototype of the DBE\_XML.XML\_DOM\_IMPORT\_NODE function is as follows:

```
DBE_XML.XML_DOM_IMPORT_NODE(
doc_id IN RAW(13),
node_id IN RAW(13),
deep IN BOOLEAN
)
RETURNS RAW(13);
```

**Table 10-116** DBE\_XML.XML\_DOM\_IMPORT\_NODE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                           |
|-----------|---------|------------------------|--------------|---------------------------------------|
| doc_id    | RAW(13) | IN                     | No           | Document to which the node is mounted |
| node_id   | RAW(13) | IN                     | No           | Node to be imported                   |

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                   |
|-----------|---------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| deep      | BOOLEAN | IN                     | No           | Specifies whether to perform recursive import. <ul style="list-style-type: none"> <li>If the value is <b>TRUE</b>, the node and all its subnodes are imported.</li> <li>If the value is <b>FALSE</b>, the node itself is imported.</li> </ul> |

- DBE\_XML.XML\_DOM\_IS\_NULL

Checks whether the given object is null. If it is null, **TRUE** is returned. Otherwise, **FALSE** is returned.

The prototype of the DBE\_XML.XML\_DOM\_IS\_NULL function is as follows:

```
DBE_XML.XML_DOM_IS_NULL (
 id RAW(13)
)
RETURNS BOOLEAN;
```

**Table 10-117** DBE\_XML.XML\_DOM\_IS\_NULL parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_ITEM

Returns the element corresponding to the index in a list or map based on the index.

The prototype of the DBE\_XML.XML\_DOM\_ITEM function is as follows:

```
DBE_XML.XML_DOM_ITEM (
 id IN RAW(13),
 index IN INTEGER
)
RETURNS RAW(13);
```

**Table 10-118** DBE\_XML.XML\_DOM\_ITEM parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                           |
|-----------|---------|------------------------|--------------|---------------------------------------|
| id        | RAW(13) | IN                     | No           | XML DOM object                        |
| index     | INTEGER | IN                     | No           | Index of the element to be retrieved. |

- DBE\_XML.XML\_DOM\_MAKE\_ELEMENT

Returns the DOMELEMENT object after conversion.

The stored procedure prototype of DBE\_XML.XML\_DOM\_MAKE\_ELEMENT is as follows:

```
DBE_XML.XML_DOM_MAKE_ELEMENT(
 id IN RAW(13))
RETURNS RAW(13);
```

**Table 10-119** DBE\_XML.XML\_DOM\_MAKE\_ELEMENT parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object. |

- DBE\_XML.XML\_DOM\_MAKENODE

Converts the given object to the DOMNode type.

The stored procedure prototype of DBE\_XML.XML\_DOM\_MAKENODE is as follows:

```
DBE_XML.XML_DOM_MAKENODE(
 id RAW(13)
)
RETURNS DOMNODE;
```

**Table 10-120** DBE\_XML.XML\_DOM\_MAKENODE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_NEW\_DOM\_DOCUMENT\_EMPTY

Returns a new DOMDocument object.

The prototype of the DBE\_XML.XML\_DOM\_NEW\_DOM\_DOCUMENT\_EMPTY function is as follows:

```
DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY()
RETURNS RAW(13);
```

- **DBE\_XML.XML\_DOM\_NEW\_DOM\_DOCUMENT\_CLOB**

Returns a new DOMDocument instance object created from the specified CLOB type.

The prototype of the DBE\_XML.XML\_DOM\_NEW\_DOM\_DOCUMENT\_CLOB function is:

```
DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_CLOB(
 content IN CLOB
)
RETURNS RAW(13);
```

**Table 10-121** DBE\_XML.XML\_DOM\_NEW\_DOM\_DOCUMENT\_CLOB parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description          |
|-----------|------|------------------------|--------------|----------------------|
| content   | CLOB | IN                     | No           | Specified CLOB type. |

- **DBE\_XML.XML\_DOM\_NEW\_DOCUMENT\_XMLTYPE**

Returns a new DOMDocument instance object created from the specified XMLType type.

The prototype of the DBE\_XML.XML\_DOM\_NEW\_DOCUMENT\_XMLTYPE function is as follows:

```
DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE(
 content IN CLOB
)
RETURNS RAW(13);
```

**Table 10-122** DBE\_XML.XML\_DOM\_NEW\_DOCUMENT\_XMLTYPE parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description          |
|-----------|------|------------------------|--------------|----------------------|
| content   | CLOB | IN                     | No           | Specified CLOB type. |

- **DBE\_XML.XML\_DOM\_NEW\_DOCUMENT\_XMLTYPE**

Sets the attributes of a specified XML DOM object.

The stored procedure prototype of DBE\_XML.XML\_DOM\_SET\_ATTRIBUTE is as follows:

```
DBE_XML.XML_DOM_SET_ATTRIBUTEe(
 docid IN RAW(13),
 name IN VARCHAR2,
 value IN VARCHAR2
)
RETURNS VOID;
```

**Table 10-123** DBE\_XML.XML\_DOM\_SET\_ATTRIBUTE parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description    |
|-----------|----------|------------------------|--------------|----------------|
| id        | RAW(13)  | IN                     | No           | XML DOM object |
| name      | VARCHAR2 | IN                     | No           | String         |
| value     | VARCHAR2 | IN                     | No           | String.        |

- DBE\_XML.XML\_DOM\_SET\_CHARSET

Sets the character set for a DOMDocument object.

The prototype of the DBE\_XML.XML\_DOM\_SET\_CHARSET function is as follows:

```
DBE_XML.XML_DOM_SET_CHARSET(
 id IN RAW(13),
 charset IN VARCHAR2
)
RETURNS VOID;
```

**Table 10-124** DBE\_XML.XML\_DOM\_SET\_CHARSET parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description    |
|-----------|----------|------------------------|--------------|----------------|
| id        | RAW(13)  | IN                     | No           | XML DOM object |
| charset   | VARCHAR2 | IN                     | No           | Character set. |

- DBE\_XML.XML\_DOM\_SET\_DOCTYPE

Sets the external DTD of a DOMDocument object.

The prototype of the DBE\_XML.XML\_DOM\_SET\_DOCTYPE function is as follows:

```
DBE_XML.XML_DOM_SET_DOCTYPE(
 id IN RAW(13),
 dtd_name IN VARCHAR2,
 system_id IN VARCHAR2,
 public_id IN VARCHAR2
)
RETURNS VOID;
```

**Table 10-125** DBE\_XML.XML\_DOM\_SET\_DOCTYPE parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                                            |
|-----------|----------|------------------------|--------------|--------------------------------------------------------|
| id        | RAW(13)  | IN                     | No           | XML DOM object                                         |
| dtd_name  | VARCHAR2 | IN                     | No           | Name of the DOCType to be initialized                  |
| system_id | VARCHAR2 | IN                     | No           | ID of the system whose DOCType needs to be initialized |
| public_id | VARCHAR2 | IN                     | No           | Public ID of the DOCType to be initialized.            |

- DBE\_XML.XML\_DOM\_SET\_NODE\_VALUE

Sets the value of a node in the DOMNode object.

The stored procedure prototype of DBE\_XML.XML\_DOM\_SET\_NODE\_VALUE is as follows:

```
DBE_XML.XML_DOM_SET_NODE_VALUE(
id IN RAW(13),
node_value IN VARCHAR2)
RETURNS VOID;
```

**Table 10-126** DBE\_XML.XML\_DOM\_SET\_NODE\_VALUE parameters

| Parameter  | Type     | Input/Output Parameter | Can Be Empty | Description                             |
|------------|----------|------------------------|--------------|-----------------------------------------|
| id         | RAW(13)  | IN                     | No           | XML DOM object                          |
| node_value | VARCHAR2 | IN                     | No           | String to be set in the DOMNode object. |

- DBE\_XML.XML\_DOM\_WRITE\_TO\_BUFFER\_DOC

Writes the given DOMDocument object to the buffer.

The stored procedure prototype of DBE\_XML.XML\_DOM\_WRITE\_TO\_BUFFER\_DOC is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC(
id IN RAW(13))
RETURNS VARCHAR2;
```

**Table 10-127** DBE\_XML.XML\_DOM\_WRITE\_TO\_BUFFER\_DOC parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_WRITE\_TO\_BUFFER\_NODE

Writes the given DOMNode object to the buffer.

The stored procedure prototype of

DBE\_XML.XML\_DOM\_WRITE\_TO\_BUFFER\_NODE is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE(
id IN RAW(13))
RETURNS VARCHAR2;
```

**Table 10-128** DBE\_XML.XML\_DOM\_WRITE\_TO\_BUFFER\_NODE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_WRITE\_TO\_CLOB\_DOC

Writes the given DOMDocument object to a CLOB.

The stored procedure prototype of

DBE\_XML.XML\_DOM\_WRITE\_TO\_CLOB\_DOC is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC(
id IN RAW(13)
)
RETURNS VARCHAR2;
```

**Table 10-129** DBE\_XML.XML\_DOM\_WRITE\_TO\_CLOB\_DOC parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_WRITE\_TO\_CLOB\_NODE

Writes the given DOMNode object to a CLOB.

The stored procedure prototype of

DBE\_XML.XML\_DOM\_WRITE\_TO\_CLOB\_NODE is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE(
id IN RAW(13)
)
RETURNS CLOB;
```

**Table 10-130** DBE\_XML.XML\_DOM\_WRITE\_TO\_CLOB\_NODE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description     |
|-----------|---------|------------------------|--------------|-----------------|
| id        | RAW(13) | IN                     | Yes          | XML DOM object. |

- DBE\_XML.XML\_DOM\_WRITE\_TO\_FILE\_DOC**  
Writes an XML node to a specified file using the database character set.  
The stored procedure prototype of DBE\_XML.XML\_DOM\_WRITE\_TO\_FILE\_DOC is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_FILE_DOC(
id IN RAW(13),
file_dir IN VARCHAR2)
RETURNS VOID;

DBE_XML.XML_DOM_WRITE_TO_FILE_DOC(
id IN RAW(13),
file_dir IN VARCHAR2,
charset IN VARCHAR2)
RETURNS VOID PACKAGE
```

**Table 10-131** DBE\_XML.XML\_DOM\_WRITE\_TO\_FILE\_DOC parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description              |
|-----------|----------|------------------------|--------------|--------------------------|
| id        | RAW(13)  | IN                     | Yes          | XML DOM object           |
| file_dir  | VARCHAR2 | IN                     | No           | File to be written       |
| charset   | VARCHAR2 | IN                     | No           | Specified character set. |

- DBE\_XML.XML\_DOM\_WRITE\_TO\_FILE\_NODE**  
Writes an XML node to a specified file using the database character set.  
The stored procedure prototype of DBE\_XML.XML\_DOM\_WRITE\_TO\_FILE\_NODE is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_FILE_NODE(
id IN RAW(13),
filename IN VARCHAR2)
RETURNS VOID;
```

**Table 10-132** DBE\_XML.XML\_DOM\_WRITE\_TO\_FILE\_NODE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description    |
|-----------|---------|------------------------|--------------|----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object |

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description             |
|-----------|----------|------------------------|--------------|-------------------------|
| filename  | VARCHAR2 | IN                     | No           | Specified file address. |

- DBE\_XML.XML\_DOM\_GET\_SESSION\_TREE\_NUM**  
Queries the number of DOM trees of all types in the current session.  
The prototype of the DBE\_XML.XML\_DOM\_GET\_SESSION\_TREE\_NUM function is as follows:

```
DBE_XML.XML_DOM_GET_SESSION_TREE_NUM()
RETURNS INTEGER;
```
- DBE\_XML.XML\_DOM\_GET\_DOC\_TREES\_INFO**  
Queries the DOM tree information of the document type in the current session, such as the memory usage.  
The prototype of the DBE\_XML.XML\_DOM\_GET\_DOC\_TREES\_INFO function is as follows:

```
DBE_XML.XML_DOM_GET_DOC_TREES_INFO()
RETURNS VARCHAR2;
```
- DBE\_XML.XML\_DOM\_GET\_DETAIL\_DOC\_TREE\_INFO**  
Queries the number of subnodes of each type in the specified document.  
The prototype of the DBE\_XML.XML\_DOM\_GET\_DETAIL\_DOC\_TREE\_INFO function is as follows:

```
DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO(
id IN RAW(13))
RETURNS VARCHAR2;
```

**Table 10-133** DBE\_XML.XML\_DOM\_GET\_DETAIL\_DOC\_TREE\_INFO parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description    |
|-----------|---------|------------------------|--------------|----------------|
| id        | RAW(13) | IN                     | No           | XML DOM object |

## 10.11.2 Secondary Encapsulation APIs (Recommended)

### 10.11.2.1 DBE\_LOB

#### APIs

**Table 10-134** lists all APIs supported by the **DBE\_LOB** package.

 NOTE

- In database A, the byte content of the space is 00. However, in GaussDB, the byte content corresponding to the space is ASCII code 32.
- In a distributed environment, the maximum size of a CLOB, BLOB, and BFILE is 1 GB.
- LOBMAXSIZE supports a maximum of 1073741771 bytes.

**Table 10-134** DBE\_LOB

| API                                      | Description                                                                                                            |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_LOB.GET_LENGTH</a>       | Obtains and returns the length of a specified LOB. The object cannot be greater than 2 GB.                             |
| <a href="#">DBE_LOB.LOB_GET_LENGTH</a>   | Obtains and returns the length of a specified LOB or BFILE object.                                                     |
| <a href="#">DBE_LOB.OPEN</a>             | Opens a LOB and returns a LOB descriptor.                                                                              |
| <a href="#">DBE_LOB.READ</a>             | Loads a part of LOB content to the buffer based on the specified length and initial position offset.                   |
| <a href="#">DBE_LOB.LOB_READ</a>         | Reads a part of LOB (including BFILE) content to the buffer based on the specified length and initial position offset. |
| <a href="#">DBE_LOB.WRITE</a>            | Copies content in the buffer to a LOB based on the specified length and initial position offset.                       |
| <a href="#">DBE_LOB.WRITE_APPEND</a>     | Copies content in the buffer to the end part of a LOB based on the specified length.                                   |
| <a href="#">DBE_LOB.LOB_WRITE_APPEND</a> | Copies content in the buffer to the end part of a LOB based on the specified length.                                   |
| <a href="#">DBE_LOB.COPY</a>             | Copies content in a LOB to another LOB based on the specified length and initial position offset.                      |
| <a href="#">DBE_LOB.LOB_COPY</a>         | Copies content in a LOB to another LOB based on the specified length and initial position offset.                      |
| <a href="#">DBE_LOB.ERASE</a>            | Deletes content in a LOB (less than or equal to 1 GB) based on the specified length and initial position offset.       |
| <a href="#">DBE_LOB.LOB_ERASE</a>        | Deletes content in a LOB based on the specified length and initial position offset.                                    |
| <a href="#">DBE_LOB.CLOSE</a>            | Closes a LOB descriptor.                                                                                               |
| <a href="#">DBE_LOB.MATCH</a>            | Returns the position of the <i>M</i> th occurrence of a character string in a LOB.                                     |
| <a href="#">DBE_LOB.COMPARE</a>          | Compares two LOBs (including BFILE objects) or a certain part of two LOBs.                                             |
| <a href="#">DBE_LOB.SUBSTR</a>           | Reads a LOB substring and returns the read substring.                                                                  |

| API                                        | Description                                                                                                                                                    |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_LOB.LOB_SUBSTR</a>         | Reads a LOB or BFILE substring and returns the read substring.                                                                                                 |
| <a href="#">DBE_LOB.STRIP</a>              | Truncates the LOB of a specified length. After the execution is complete, the length of the LOB is set to the length specified by the <b>newlen</b> parameter. |
| <a href="#">DBE_LOB.LOB_STRIP</a>          | Truncates the LOB of a specified length. After the execution is complete, the length of the LOB is set to the length specified by the <b>newlen</b> parameter. |
| <a href="#">DBE_LOB.CREATE_TEMPORARY</a>   | Creates a temporary BLOB or CLOB.                                                                                                                              |
| <a href="#">DBE_LOB.APPEND</a>             | Adds the content of a LOB to another LOB.                                                                                                                      |
| <a href="#">DBE_LOB.LOB_APPEND</a>         | Adds the content of a LOB to another LOB.                                                                                                                      |
| <a href="#">DBE_LOB.FREETEMPORARY</a>      | Deletes a temporary BLOB or CLOB.                                                                                                                              |
| <a href="#">DBE_LOB.FILEOPEN</a>           | Opens a database BFILE file and returns its file descriptor.                                                                                                   |
| <a href="#">DBE_LOB.FILECLOSE</a>          | Closes a BFILE file that is opened by <b>FILEOPEN</b> .                                                                                                        |
| <a href="#">DBE_LOB.BFILEOPEN</a>          | Opens a database BFILE file.                                                                                                                                   |
| <a href="#">DBE_LOB.BFILECLOSE</a>         | Closes a BFILE file that is opened by <b>BFILEOPEN</b> .                                                                                                       |
| <a href="#">DBE_LOB.LOADFROMFILE</a>       | Loads the database BFILE file of the specified length in the specified location to the BLOB in the specified location.                                         |
| <a href="#">DBE_LOB.LOADFROMBFILE</a>      | Loads the database BFILE file of the specified length in the specified location to the LOB in the specified location.                                          |
| <a href="#">DBE_LOB.LOADBLOBFROMFILE</a>   | Loads an external database file of the specified length in the specified location to a BLOB (less than or equal to 1 GB) in a specified location.              |
| <a href="#">DBE_LOB.LOADBLOBFROMFIL...</a> | Loads the database BFILE file of the specified length in the specified location to the BLOB in the specified location.                                         |
| <a href="#">DBE_LOB.LOADCLOBFROMFILE</a>   | Loads an external database file of the specified length in the specified location to a CLOB (less than or equal to 1 GB) in a specified location.              |
| <a href="#">DBE_LOB.LOADCLOBFROMFIL...</a> | Loads the database BFILE file of the specified length in the specified location to the CLOB in the specified location.                                         |
| <a href="#">DBE_LOB.CONVERTTOBLOB</a>      | Converts a CLOB file to a BLOB file (less than or equal to 1 GB).                                                                                              |

| API                                          | Description                                                                                                                                                                                                 |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_LOB.CONVERTTOCLOB</a>        | Converts a BLOB file to a CLOB file (less than or equal to 1 GB).                                                                                                                                           |
| <a href="#">DBE_LOB.LOB_CONVERTTOBLO...</a>  | Converts a CLOB file to a BLOB file.                                                                                                                                                                        |
| <a href="#">DBE_LOB.LOB_CONVERTTTOCLO...</a> | Converts a BLOB file to a CLOB file.                                                                                                                                                                        |
| <a href="#">DBE_LOB.GETCHUNKSIZE</a>         | Obtains the maximum size of LOB data that can be stored in the chunk structure in the database.                                                                                                             |
| <a href="#">DBE_LOB.LOB_WRITE</a>            | Reads the specified length of the source object from the start position, writes the content to the specified offset position of the target LOB, overrides the original content, and returns the target LOB. |
| <a href="#">DBE_LOB.BFILENAME</a>            | Constructs and returns the DBE_LOB.BFILE object based on the directory and file name.                                                                                                                       |

- [DBE\\_LOB.GET\\_LENGTH](#)

Obtains and returns the size of a specified LOB. The maximum size of the object is 2 GB.

The prototype of the [DBE\\_LOB.GET\\_LENGTH](#) function is as follows:

```
DBE_LOB.GET_LENGTH (
 blob_obj IN BLOB)
RETURN INTEGER;

DBE_LOB.GET_LENGTH (
 clob_obj IN CLOB)
RETURN INTEGER;
```

**Table 10-135** [DBE\\_LOB.GET\\_LENGTH](#) parameters

| Parameter             | Description                              |
|-----------------------|------------------------------------------|
| blob_obj/<br>clob_obj | BLOB/CLOB whose length is to be obtained |

- [DBE\\_LOB.LOB\\_GET\\_LENGTH](#)

Obtains and returns the length of a specified LOB or BFILE file. The maximum size of the object is 32 TB.

The prototype of the [DBE\\_LOB.LOB\\_GET\\_LENGTH](#) function is as follows:

```
DBE_LOB.LOB_GET_LENGTH (
 blob_obj IN BLOB)
RETURN BIGINT;

DBE_LOB.LOB_GET_LENGTH (
 clob_obj IN CLOB)
RETURN BIGINT;

DBE_LOB.LOB_GET_LENGTH (
```

```
bfile IN DBE_LOB.BFILE)
RETURN BIGINT;
```

**Table 10-136** DBE\_LOB.LOB\_GET\_LENGTH parameters

| Parameter                   | Description                                    |
|-----------------------------|------------------------------------------------|
| blob_obj/<br>clob_obj/bfile | BLOB/CLOB/BFILE whose length is to be obtained |

- DBE\_LOB.OPEN

Opens a LOB and returns a LOB descriptor. This procedure is meaningless and is used only for compatibility.

The prototype of the DBE\_LOB.OPEN function is as follows:

```
DBE_LOB.OPEN (
 lob INOUT BLOB);

DBE_LOB.OPEN (
 lob INOUT CLOB);

DBE_LOB.OPEN (
 bfile INOUT DBE_LOB.BFILE,
 open_mode IN TEXT DEFAULT 'null');
```

**Table 10-137** DBE\_LOB.OPEN parameters

| Parameter | Description                                               |
|-----------|-----------------------------------------------------------|
| lob/bfile | Opened BLOB, CLOB or BFILE.                               |
| open_mode | Operation mode. Currently, the range is [R,W,A,RB,WB,AB]. |

- DBE\_LOB.READ

Reads a part of LOB content to the output buffer based on the specified length and initial position offset.

The prototype of the DBE\_LOB.READ function is as follows:

```
DBE_LOB.READ (
 blob_obj IN BLOB,
 amount IN INTEGER,
 off_set IN INTEGER,
 out_put OUT RAW);

DBE_LOB.READ (
 clob_obj IN CLOB,
 amount IN INTEGER,
 off_set IN INTEGER,
 out_put OUT VARCHAR2);
```

**Table 10-138** DBE\_LOB.READ parameters

| Parameter             | Description          |
|-----------------------|----------------------|
| blob_obj/<br>clob_obj | BLOB/CLOB to be read |

| Parameter | Description                                                                                                                                                                                                                 |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| amount    | Length of read content<br><b>NOTE</b><br>If the length to read is less than 1 or greater than 32767, an error is reported.                                                                                                  |
| off_set   | Start position for reading the LOB content, that is, the offset bytes to initial position of the LOB content. If the offset is less than 1 or greater than the LOB length, an error is reported. The initial position is 1. |
| out_put   | Target buffer for storing the read LOB content                                                                                                                                                                              |

- DBE\_LOB.LOB\_READ

Reads a part of LOB/BFILE content to the output buffer based on the specified length and initial position offset.

The prototype of the DBE\_LOB.LOB\_READ function is as follows:

```
DBE_LOB.LOB_READ(
 blob_obj IN BLOB,
 amount INOUT BIGINT,
 off_set IN BIGINT,
 out_put OUT RAW);

DBE_LOB.LOB_READ(
 clob_obj IN CLOB,
 amount INOUT BIGINT,
 off_set IN BIGINT,
 out_put OUT VARCHAR2);

DBE_LOB.LOB_READ(
 bfile IN DBE_LOB.BFILE,
 amount INOUT BIGINT,
 off_set IN BIGINT,
 out_put OUT RAW);
```

**Table 10-139** DBE\_LOB.LOB\_READ parameters

| Parameter                   | Description                                                                                                                                                                                                                 |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blob_obj/<br>clob_obj/bfile | BLOB/CLOB/BFILE object (can be greater than 1 GB) to be read                                                                                                                                                                |
| amount                      | Length to read as the IN parameter, or actual read length as the OUT parameter.<br><b>NOTE</b><br>If the length to read is less than 1 or greater than 32767, an error is reported.                                         |
| off_set                     | Start position for reading the LOB content, that is, the offset bytes to initial position of the LOB content. If the offset is less than 1 or greater than the LOB length, an error is reported. The initial position is 1. |
| out_put                     | Target buffer for storing the read LOB content                                                                                                                                                                              |

- DBE\_LOB.WRITE

Writes content in the source to a LOB based on the specified length and initial position.

The prototype of the DBE\_LOB.WRITE function is as follows:

```
DBE_LOB.WRITE (
 blob_obj INOUT BLOB,
 amount IN INTEGER,
 off_set IN INTEGER,
 source IN RAW);

DBE_LOB.WRITE (
 clob_obj INOUT CLOB,
 amount IN INTEGER,
 off_set IN INTEGER,
 source IN VARCHAR2);
```

**Table 10-140** DBE\_LOB.WRITE parameters

| Parameter             | Description                                                                                                                                                                                                                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blob_obj/<br>clob_obj | BLOB/CLOB to which the buffer content is written                                                                                                                                                                                                                                                                                      |
| amount                | Length to write, up to 32767 characters.<br><b>NOTE</b><br>If the length to write is less than 1 or greater than the length of the content to be written, an error is reported.                                                                                                                                                       |
| off_set               | Start position for writing content to the target LOB, that is, the offset bytes to the initial position of LOB content.<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <b>LOBMAXSIZE</b> , an error is reported. The initial position is 1, and the maximum value is the maximum length of the LOB type. |
| source                | Content to be written                                                                                                                                                                                                                                                                                                                 |

- DBE\_LOB.WRITE\_APPEND

Writes content in the source object to the end part of a LOB based on the specified length.

The prototype of the DBE\_LOB.WRITE\_APPEND function is as follows:

```
DBE_LOB.WRITE_APPEND (
 blob_obj INOUT BLOB,
 amount IN INTEGER,
 source_obj IN RAW);

DBE_LOB.WRITE_APPEND (
 clob_obj INOUT CLOB,
 amount IN INTEGER,
 source_obj IN VARCHAR2);
```

**Table 10-141** DBE\_LOB.WRITE\_APPEND parameters

| Parameter             | Description                                      |
|-----------------------|--------------------------------------------------|
| blob_obj/<br>clob_obj | BLOB/CLOB to which the buffer content is written |

| Parameter  | Description                                                                                                                                                                     |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| amount     | Length to write, up to 32767 characters.<br><b>NOTE</b><br>If the length to write is less than 1 or greater than the length of the content to be written, an error is reported. |
| source_obj | Content to be written                                                                                                                                                           |

- DBE\_LOB.LOB\_WRITE\_APPEND

Writes content in the source object to the end part of a LOB based on the specified length.

The prototype of the DBE\_LOB.LOB\_WRITE\_APPEND function is as follows:

```
DBE_LOB.LOB_WRITE_APPEND(
 blob_obj INOUT BLOB,
 amount IN INTEGER,
 source_obj IN RAW);
```

```
DBE_LOB.LOB_WRITE_APPEND (
 clob_obj INOUT CLOB,
 amount IN INTEGER,
 source_obj IN VARCHAR2);
```

**Table 10-142** DBE\_LOB.LOB\_WRITE\_APPEND parameters

| Parameter             | Description                                                                                                                                                                     |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blob_obj/<br>clob_obj | BLOB/CLOB to which the buffer content is written                                                                                                                                |
| amount                | Length to write, up to 32767 characters.<br><b>NOTE</b><br>If the length to write is less than 1 or greater than the length of the content to be written, an error is reported. |
| source_obj            | Content to be written                                                                                                                                                           |

- DBE\_LOB.COPY

Copies content in a LOB to another LOB based on the specified length and initial position offset.

The prototype of the DBE\_LOB.COPY function is as follows:

```
DBE_LOB.COPY (
 dest_lob INOUT BLOB,
 src_lob IN BLOB,
 len IN INTEGER,
 dest_start IN INTEGER DEFAULT 1,
 src_start IN INTEGER DEFAULT 1);
```

**Table 10-143** DBE\_LOB.COPY parameters

| Parameter | Description                                       |
|-----------|---------------------------------------------------|
| dest_lob  | LOB to which the buffer content is to be pasted   |
| src_lob   | LOB from which the buffer content is to be copied |

| Parameter  | Description                                                                                                                                             |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| len        | Length to copy                                                                                                                                          |
| dest_start | Start position for pasting the buffer content to the target LOB ( <b>dest_lob</b> ), that is, the offset bytes to the initial position of LOB content.  |
| src_start  | Start position for copying the buffer content from the source LOB ( <b>src_lob</b> ), that is, the offset bytes to the initial position of LOB content. |

- DBE\_LOB.LOB\_COPY

Copies content in a LOB to another LOB based on the specified length and initial position offset.

The prototype of the DBE\_LOB.LOB\_COPY function is as follows:

```
DBE_LOB.LOB_COPY(
 blob_obj INOUT BLOB,
 source_obj IN BLOB,
 amount IN BIGINT,
 dest_offset IN BIGINT DEFAULT 1,
 src_offset IN BIGINT DEFAULT 1);

DBE_LOB.LOB_COPY(
 clob_obj INOUT CLOB,
 source_obj IN CLOB,
 amount IN BIGINT,
 dest_offset IN BIGINT DEFAULT 1,
 src_offset IN BIGINT DEFAULT 1);
```

**Table 10-144** DBE\_LOB.LOB\_COPY parameters

| Parameter             | Description                                                                                                                                                                                                                                                                                                           |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blob_obj/<br>clob_obj | LOB to which the buffer content is to be pasted                                                                                                                                                                                                                                                                       |
| source_obj            | LOB from which the buffer content is to be copied                                                                                                                                                                                                                                                                     |
| amount                | Length of the copied data<br><b>NOTE</b><br>If the length to copy is less than 1 or greater than the value of <b>LOBMAXSIZE</b> , an error is reported.                                                                                                                                                               |
| dest_offset           | Start position for pasting the buffer content to the target LOB, that is, the offset bytes/characters to the initial position of LOB content. The unit is byte for BLOB and character for CLOB.<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <b>LOBMAXSIZE</b> , an error is reported. |

| Parameter  | Description                                                                                                                                                                                                                                                  |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| src_offset | Start position for copying the content from the source object, that is, the offset bytes to the initial position of LOB content. The unit is byte for BLOB and character for CLOB.<br><br><b>NOTE</b><br>If the offset is less than 1, an error is reported. |

- DBE\_LOB.ERASE

Deletes the content in a BLOB (not greater than 1 GB) based on the specified length and initial position offset. The bytes of the deleted part in the BLOB are filled with 0.

The prototype of the DBE\_LOB.ERASE function is as follows:

```
DBE_LOB.ERASE (
 blob_obj INOUT BLOB,
 amount INOUT INTEGER,
 off_set IN INTEGER DEFAULT 1);
```

**Table 10-145** DBE\_LOB.ERASE parameters

| Parameter | Description                                                                                                                                                                                                                                                                          |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blob_obj  | LOB whose content is to be deleted as the IN parameter, or LOB with specified content deleted as the OUT parameter. If this parameter is left empty, an error is reported.                                                                                                           |
| amount    | Length (in bytes for BLOBs) to delete as the IN parameter, or actual length deleted as the OUT parameter.<br><br><b>NOTE</b><br>If the length to delete is less than 1 or this parameter is left empty, an error is reported.                                                        |
| off_set   | Start position from which LOB content is to be deleted, that is, the number of offset bytes to the initial position of LOB content. The value cannot be greater than 1 GB.<br><br><b>NOTE</b><br>If the offset is less than 1 or this parameter is left empty, an error is reported. |

- DBE\_LOB.LOB\_ERASE

Deletes the content in the LOB based on the specified length and initial position offset. The bytes of the deleted part in the BLOB are filled with 0, and the characters of the deleted part in the CLOB are filled with spaces. The LOB can be greater than 1 GB and the maximum size is 32 TB.

The prototype of the DBE\_LOB.LOB\_ERASE function is as follows:

```
DBE_LOB.LOB_ERASE (
 blob_obj INOUT BLOB,
 amount INOUT BIGINT,
 off_set IN BIGINT DEFAULT 1);
```

```
DBE_LOB.LOB_ERASE (
```

```
clob_obj INOUT CLOB,
amount INOUT BIGINT,
off_set IN BIGINT DEFAULT 1);
```

**Table 10-146** DBE\_LOB.LOB\_ERASE parameters

| Parameter             | Description                                                                                                                                                                                                                                                                                                                       |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blob_obj/<br>clob_obj | LOB whose content is to be deleted as the IN parameter, or LOB with specified content deleted as the OUT parameter. If this parameter is left empty, an error is reported.                                                                                                                                                        |
| amount                | Length (BLOB in bytes and CLOB in characters) to delete as the IN parameter, or actual length deleted as the OUT parameter.<br><b>NOTE</b><br>If the length to delete is less than 1 or this parameter is left empty, an error is reported.                                                                                       |
| off_set               | Start position from which the LOB content is to be deleted, that is, the number of bytes relative to the start position of the BLOB content or the number of characters relative to the start position of the CLOB content.<br><b>NOTE</b><br>If the offset is less than 1 or this parameter is left empty, an error is reported. |

- DBE\_LOB.CLOSE

Closes the LOB descriptor that has been opened.

The prototype of the DBE\_LOB.CLOSE function is as follows:

```
DBE_LOB.CLOSE(
lob IN BLOB);

DBE_LOB.CLOSE (
lob IN CLOB);

DBE_LOB.CLOSE (
file IN INTEGER);
```

**Table 10-147** DBE\_LOB.CLOSE parameters

| Parameter | Description                                                |
|-----------|------------------------------------------------------------|
| lob/file  | BLOB/CLOB/File object whose LOB descriptor is to be closed |

- DBE\_LOB.MATCH

Returns the *M*th occurrence position of a string in a LOB or BFILE file. **NULL** is returned for an invalid input. The LOB or BFILE file can be greater than 1 GB, up to 32 TB.

The prototype of the DBE\_LOB.MATCH function is as follows:

```
DBE_LOB.MATCH(
blob_obj IN BLOB,
blob_obj2 IN RAW,
```

```

 beg_index IN BIGINT DEFAULT 1,
 occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;

DBE_LOB.MATCH(
 clob_obj IN CLOB,
 clob_obj2 IN VARCHAR2,
 beg_index IN BIGINT DEFAULT 1,
 occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;

DBE_LOB.MATCH(
 bfile IN DBE_LOB.BFILE,
 blob_obj2 IN RAW,
 beg_index IN BIGINT DEFAULT 1,
 occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;

```

**Table 10-148** DBE\_LOB.MATCH parameters

| Parameter                       | Description                                                                                                                                                                                                                                                                                    |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blob_obj/<br>clob_obj/<br>bfile | BLOB/CLOB descriptor to be searched for, or BFILE (which must be opened using <b>DBE_LOB.BFILEOPEN</b> first). If this parameter is left blank, null is returned.                                                                                                                              |
| blob_obj<br>2/<br>clob_obj2     | Pattern to match. It is RAW for BLOB/BFILE objects and VARCHAR2 for CLOBs. If this parameter is left empty, null is returned.                                                                                                                                                                  |
| beg_index                       | Absolute offset (in bytes) for BLOB/BFILE objects, or offset (in characters) for CLOBs. The start position for matching is 1.<br><b>NOTE</b><br>The value ranges from 1 to <i>LOBMAXSIZE</i> . If a value out of the range is input, null is returned.                                         |
| occur_index                     | Number of pattern matching times. The minimum value is 1.<br><b>NOTE</b><br>If the value is greater than the maximum number of times that the pattern string can be matched in the LOB, <b>0</b> is returned. If the value is out of the range from 1 to <i>LOBMAXSIZE</i> , null is returned. |

- **DBE\_LOB.COMPARE**

Compares LOBs or BFILE objects.

- If the compared objects are equal, **0** is returned. Otherwise, a non-zero value is returned.
- If the first LOB is smaller than the second, **-1** is returned. If the first LOB is larger than the second, **1** is returned.
- If any of the *len*, *start1*, and *start2* parameters is invalid, **NULL** is returned. The valid offset range is 1 to *LOBMAXSIZE*.
- If both the values of **start\_pos1** and **start\_pos2** exceed the LOB/BFILE length, **0** is returned.

The prototype of the DBE\_LOB.COMPARE function is as follows:

```

DBE_LOB.COMPARE (
 lob1 IN BLOB,
 lob2 IN BLOB,

```

```

len IN BIGINT DEFAULT 1073741312,
start_pos1 IN BIGINT DEFAULT 1,
start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;

DBE_LOB.COMPARE (
lob1 IN CLOB,
lob2 IN CLOB,
len IN BIGINT DEFAULT 1073741312,
start_pos1 IN BIGINT DEFAULT 1,
start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;

DBE_LOB.COMPARE (
file1 IN DBE_LOB.BFILE,
file2 IN DBE_LOB.BFILE,
len IN BIGINT DEFAULT 1073741312,
start_pos1 IN BIGINT DEFAULT 1,
start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;

```

**Table 10-149** DBE\_LOB.COMPARE parameters

| Parameter  | Description                                                                                                                     |
|------------|---------------------------------------------------------------------------------------------------------------------------------|
| lob1/file1 | First BLOB/CLOB/BFILE to be compared (which must be opened using <b>DBE_LOB.BFILEOPEN</b> first).                               |
| lob2/file2 | Second BLOB/CLOB/BFILE to be compared (which must be opened using <b>DBE_LOB.BFILEOPEN</b> first).                              |
| len        | Number of characters or bytes to be compared. The default value is <b>1073741312</b> .                                          |
| start_pos1 | Offset of the first LOB descriptor. The initial position is 1, and the maximum value is the maximum length of the LOB content.  |
| start_pos2 | Offset of the second LOB descriptor. The initial position is 1, and the maximum value is the maximum length of the LOB content. |

- **DBE\_LOB.SUBSTR**

Reads a LOB substring and returns the read substring.

The prototype of the DBE\_LOB.SUBSTR function is as follows:

```

DBE_LOB.SUBSTR(
lob_loc IN BLOB,
amount IN INTEGER DEFAULT 32767,
off_set IN INTEGER DEFAULT 1)
RETURN RAW;

DBE_LOB.SUBSTR(
lob_loc IN CLOB,
amount IN INTEGER DEFAULT 32767,
off_set IN INTEGER DEFAULT 1)
RETURN VARCHAR2;

```

**Table 10-150** DBE\_LOB.SUBSTR parameters

| Parameter | Description                                                                                                                                                                      |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lob_loc   | LOB descriptor whose substring is to be read. For BLOBs, the return value is of the RAW type. For CLOBs, the return value is of the VARCHAR2 type.                               |
| amount    | Number of bytes or characters to be read.<br><b>NOTE</b><br>The value ranges from 1 to 32767. If the value exceeds the range, null is returned.                                  |
| off_set   | Number of characters or bytes offset from the start position.<br><b>NOTE</b><br>The value ranges from 1 to <i>LOBMAXSIZE</i> . If the value exceeds the range, null is returned. |

- DBE\_LOB.LOB\_SUBSTR

Reads a LOB or BFILE substring and returns the read substring. The LOB or BFILE file can be greater than 1 GB, up to 32 TB.

The prototype of the DBE\_LOB.LOB\_SUBSTR function is as follows:

```
DBE_LOB.LOB_SUBSTR(
 lob_loc IN BLOB,
 amount IN INTEGER DEFAULT 32767,
 off_set IN BIGINT DEFAULT 1)
RETURN RAW;

DBE_LOB.LOB_SUBSTR(
 lob_loc IN CLOB,
 amount IN INTEGER DEFAULT 32767,
 off_set IN BIGINT DEFAULT 1)
RETURN VARCHAR2;

DBE_LOB.LOB_SUBSTR(
 bfile IN DBE_LOB.BFILE,
 amount IN INTEGER DEFAULT 32767,
 off_set IN BIGINT DEFAULT 1)
RETURN RAW;
```

**Table 10-151** DBE\_LOB.LOB\_SUBSTR parameters

| Parameter         | Description                                                                                                                                                                                                                       |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lob_loc/<br>bfile | LOB descriptor or BFILE file whose substring is to be read. The file must be opened by DBE_LOB.BFILEOPEN first. For BLOBs/ BFILE files, the return value is of the RAW type. For CLOBs, the return value is of the VARCHAR2 type. |
| amount            | Number of bytes or characters to be read.<br><b>NOTE</b><br>The value ranges from 1 to 32767. If the value exceeds the range, null is returned.                                                                                   |
| off_set           | Number of characters or bytes offset from the start position.<br><b>NOTE</b><br>The value ranges from 1 to <i>LOBMAXSIZE</i> . If the value exceeds the range, null is returned.                                                  |

- **DBE\_LOB.STRIP**  
This stored procedure truncates the LOB of a specified length. After this stored procedure is executed, the length of the LOB is set to the length specified by the **newlen** parameter.

The prototype of the DBE\_LOB.STRIP function is as follows:

```
DBE_LOB.STRIP(
 lob_loc INOUT BLOB,
 newlen IN INTEGER);

DBE_LOB.STRIP(
 lob_loc INOUT CLOB,
 newlen IN INTEGER);
```

**Table 10-152** DBE\_LOB.STRIP parameters

| Parameter | Description                                                                                                                       |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------|
| lob_loc   | LOB to read as the IN parameter, or truncated object as the OUT parameter. If this parameter is left empty, an error is reported. |
| newlen    | New length after truncation, in bytes for BLOBs or in characters for CLOBs.                                                       |

- **DBE\_LOB.LOB\_STRIP**  
Truncates a LOB based on a specified length. After this stored procedure is executed, the length of the LOB is set to the length specified by the **newlen** parameter. The LOB can be greater than 1 GB, up to 32 TB.

The prototype of the DBE\_LOB.LOB\_STRIP function is as follows:

```
DBE_LOB.LOB_STRIP(
 lob_loc INOUT BLOB,
 newlen IN BIGINT);

DBE_LOB.LOB_STRIP(
 lob_loc INOUT CLOB,
 newlen IN BIGINT);
```

**Table 10-153** DBE\_LOB.LOB\_STRIP parameters

| Parameter | Description                                                                                                                                                                                                     |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lob_loc   | LOB to read as the IN parameter, or truncated object as the OUT parameter. If this parameter is left empty, an error is reported.                                                                               |
| newlen    | New length after truncation, in bytes for BLOBs or in characters for CLOBs.<br><b>NOTE</b><br>If the value is less than 1, null is returned. If the value is greater than the LOB length, an error is reported. |

- **DBE\_LOB.CREATE\_TEMPORARY**  
Creates a temporary BLOB or CLOB. This API is used only for syntax compatibility.

The prototype of the DBE\_LOB.CREATE\_TEMPORARY function is as follows:

```
DBE_LOB.CREATE_TEMPORARY (
 lob_loc INOUT BLOB,
 cache IN BOOLEAN,
 dur IN INTEGER DEFAULT 10);

DBE_LOB.CREATE_TEMPORARY (
 lob_loc INOUT CLOB,
 cache IN BOOLEAN,
 dur IN INTEGER DEFAULT 10);
```

**Table 10-154** DBE\_LOB.CREATE\_TEMPORARY parameters

| Parameter | Description                         |
|-----------|-------------------------------------|
| lob_loc   | LOB descriptor                      |
| cache     | Used only for syntax compatibility. |
| dur       | Used only for syntax compatibility. |

- DBE\_LOB.APPEND

Appends *source\_obj* to the end of the target LOB.

The prototype of the DBE\_LOB.APPEND function is as follows:

```
DBE_LOB.APPEND (
 blob_obj INOUT BLOB,
 source_obj IN BLOB);

DBE_LOB.APPEND (
 clob_obj INOUT CLOB,
 source_obj IN CLOB);
```

**Table 10-155** DBE\_LOB.APPEND parameters

| Parameter             | Description                                            |
|-----------------------|--------------------------------------------------------|
| blob_obj/<br>clob_obj | BLOB/CLOB to which the buffer content is to be written |
| source_obj            | BLOB/CLOB from which the buffer content is to be read  |

- DBE\_LOB.LOB\_APPEND

Appends *source\_obj* to the end of the target LOB.

The prototype of the DBE\_LOB.LOB\_APPEND function is as follows:

```
DBE_LOB.LOB_APPEND(
 blob_obj INOUT BLOB,
 source_obj IN BLOB);

DBE_LOB.LOB_APPEND(
 clob_obj INOUT CLOB,
 source_obj IN CLOB);
```

**Table 10-156** DBE\_LOB.LOB\_APPEND parameters

| Parameter             | Description                                            |
|-----------------------|--------------------------------------------------------|
| blob_obj/<br>clob_obj | BLOB/CLOB to which the buffer content is to be written |
| source_obj            | BLOB/CLOB from which the buffer content is to be read  |

- DBE\_LOB.FREETEMPORARY

Frees LOB files created by CREATE\_TEMPORARY.

The prototype of the DBE\_LOB.FREETEMPORARY function is as follows:

```
DBE_LOB.FREETEMPORARY (
 blob INOUT BLOB);
```

```
DBE_LOB.FREETEMPORARY (
 clob INOUT CLOB);
```

**Table 10-157** DBE\_LOB.FREETEMPORARY parameters

| Parameter | Description            |
|-----------|------------------------|
| blob/clob | BLOB/CLOB to be freed. |

- DBE\_LOB.FILEOPEN

Opens an external database BFILE file and returns its file descriptor. A maximum of 10 BFILE files can be opened in a session.

The BFILE type is defined as follows:

```
DBE_LOB.BFILE (
 directory TEXT,
 filename TEXT,
 fd INTEGER);
```

The prototype of the DBE\_LOB.FILEOPEN function is as follows:

```
DBE_LOB.FILEOPEN (
 bfile IN DBE_LOB.BFILE,
 open_mode IN TEXT)
RETURN INTEGER;
```

**Table 10-158** DBE\_LOB.FILEOPEN parameters

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| bfile     | <p>External database file to be opened. For a BFILE file, this parameter specifies the file path, file name, and file descriptor.</p> <p><b>NOTE</b><br/>The <i>file</i> variable contains the location of the file directory <i>directory</i> and the file name <i>filename</i>.</p> <ul style="list-style-type: none"> <li>• Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>• When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to read and write files in the file path specified by <b>safe_data_path</b>.</li> <li>• File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the <b>OPEN</b> function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).</li> </ul> |
| open_mode | File open mode, which can only be <b>r</b> (that is, read). An error is reported in other modes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

- DBE\_LOB.FILECLOSE

Closes an external BFILE file.

The prototype of the DBE\_LOB.FILECLOSE function is as follows:

```
DBE_LOB.FILECLOSE (
 file IN INTEGER);
```

**Table 10-159** DBE\_LOB.FILECLOSE parameters

| Parameter | Description                                                       |
|-----------|-------------------------------------------------------------------|
| file      | External database file to be closed (that is opened by FILEOPEN). |

- DBE\_LOB.BFILEOPEN

Opens an external database BFILE file. A maximum of 10 BFILE files can be opened in a session.

The prototype of DBE\_LOB.BFILEOPEN is as follows:

```
DBE_LOB.BFILEOPEN (
 bfile INOUT DBE_LOB.BFILE,
 open_mode IN TEXT DEFAULT 'R');
```

**Table 10-160** DBE\_LOB.BFILEOPEN parameters

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| bfile     | <p>Opened database BFILE file as the INOUT parameter.</p> <p><b>NOTE</b><br/>The <i>bfile</i> variable contains the location of the file directory <i>directory</i> and the file name <i>filename</i>.</p> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to read and write files in the file path specified by <b>safe_data_path</b>.</li> <li>File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the <b>OPEN</b> function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).</li> </ul> |
| open_mode | File open mode, which can only be <b>r</b> (that is, read). An error is reported in other modes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

**Examples**

```
-- Obtain the substring of the BFILE.
DECLARE
bfile dbe_lob.bfile;
BEGIN
bfile = DBE_LOB.BFILENAME(dir_name, file_name); -- Obtain the corresponding BFILE.
DBE_LOB.bfileopen(bfile, 'r'); -- Open the BFILE.
RAISE NOTICE 'res:%', DBE_LOB.lob_substr(bfile, 10, 1); -- Obtain and print the substring.
DBE_LOB.bfileclose(bfile);-- Close the BFILE.
END;
/
```

- **DBE\_LOB.BFILECLOSE**

Closes an external database BFILE file.

The prototype of DBE\_LOB.BFILECLOSE is as follows:

```
DBE_LOB.BFILECLOSE (
 bfile INOUT DBE_LOB.BFILE);
```

**Table 10-161** DBE\_LOB.BFILECLOSE parameters

| Parameter | Description                                        |
|-----------|----------------------------------------------------|
| bfile     | Closed database BFILE file as the INOUT parameter. |

- **DBE\_LOB.LOADFROMFILE**

Loads an external BFILE file to a BLOB and returns the object of the RAW type.

The prototype of the DBE\_LOB.LOADFROMFILE function is as follows:

```
DBE_LOB.LOADFROMFILE (
 dest_lob IN BLOB,
 src_file IN INTEGER,
 amount IN INTEGER,
```

```
dest_offset IN INTEGER,
src_offset IN INTEGER)
RETURN RAW;
```

**Table 10-162** DBE\_LOB.LOADFROMFILE parameters

| Parameter   | Description                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_lob    | Target BLOB. The BFILE file will be loaded to the specified offset position of the BLOB.                                                               |
| src_bfile   | Source BFILE file to be read.                                                                                                                          |
| amount      | Length of the content to be read from the BFILE file.<br><b>NOTE</b><br>If the length is less than 1 or greater than 32767, an error is reported.      |
| dest_offset | Offset length of the BLOB.<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.       |
| src_offset  | Offset length of the BFILE file.<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported. |

- DBE\_LOB.LOADFROMBFILE

Loads an external BFILE file to a LOB.

The prototype of the DBE\_LOB.LOADFROMBFILE function is as follows:

```
DBE_LOB.LOADFROMBFILE (
 dest_lob INOUT BLOB,
 src_file IN DBE_LOB.BFILE,
 amount IN BIGINT,
 dest_offset IN BIGINT DEFAULT 1,
 src_offset IN BIGINT DEFAULT 1)
RETURN BLOB;
```

```
DBE_LOB.LOADFROMBFILE (
 dest_lob INOUT CLOB,
 src_file IN DBE_LOB.BFILE,
 amount IN BIGINT,
 dest_offset IN BIGINT DEFAULT 1,
 src_offset IN BIGINT DEFAULT 1)
RETURN CLOB;
```

**Table 10-163** DBE\_LOB.LOADFROMBFILE parameters

| Parameter | Description                                                                                                                                                                          |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_lob  | Target LOB as the INOUT parameter, to which the BFILE file will be loaded. The file must be opened by <b>DBE_LOB.BFILEOPEN</b> first. The LOB can be greater than 1 GB, up to 32 TB. |

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                        |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| src_file    | Source BFILE file to be read. The BFILE file can be greater than 1 GB, up to 32 TB.                                                                                                                                                                                                                                                                |
| amount      | Length of the content to be read from the BFILE file and to be written to the LOB.<br><b>NOTE</b><br>If the length is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                                                                                                                           |
| dest_offset | Offset length of the target LOB<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                                                                                                                                                                              |
| src_offset  | Offset length of the source BFILE file<br><b>NOTE</b> <ul style="list-style-type: none"> <li>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i>, an error is reported.</li> <li>If the sum of <i>amount</i> and <i>src_offset</i> is greater than the length of <i>src_bfile</i> plus 1, an error is reported.</li> </ul> |

- DBE\_LOB.LOADBLOBFROMFILE

Loads an external BFILE file to a BLOB and returns the object of the RAW type.

The prototype of the DBE\_LOB.LOADBLOBFROMFILE function is as follows:

```
DBE_LOB.LOADBLOBFROMFILE (
 dest_lob IN BLOB,
 src_file IN INTEGER,
 amount IN INTEGER,
 dest_offset IN INTEGER,
 src_offset IN INTEGER)
RETURN RAW;
```

**Table 10-164** DBE\_LOB.LOADBLOBFROMFILE parameters

| Parameter   | Description                                                                                                                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_lob    | Target BLOB, to which the BFILE file will be loaded.                                                                                                                                                           |
| src_file    | Source BFILE file to be read.                                                                                                                                                                                  |
| amount      | Length of the target BLOB. If the length of a file exceeds this threshold, the file will not be saved to the BLOB.<br><b>NOTE</b><br>If the length is less than 1 or greater than 32767, an error is reported. |
| dest_offset | Offset length of the BLOB.<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                                               |

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                  |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| src_offset | Offset length of the BFILE file.<br><b>NOTE</b> <ul style="list-style-type: none"> <li>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i>, an error is reported.</li> <li>If the sum of <i>amount</i> and <i>src_offset</i> is greater than the length of <i>src_bfile</i> plus 1, an error is reported.</li> </ul> |

- DBE\_LOB.LOADBLOBFROMBFILE

Loads an external BFILE file to a BLOB.

The prototype of the DBE\_LOB.LOADBLOBFROMBFILE function is as follows:

```
DBE_LOB.LOADBLOBFROMBFILE (
 dest_lob INOUT BLOB,
 src_file IN DEB_LOB.BFILE,
 amount IN BIGINT,
 dest_offset INOUT BIGINT,
 src_offset INOUT BIGINT)
```

**Table 10-165** DBE\_LOB.LOADBLOBFROMBFILE parameters

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_lob    | Target BLOB as the INOUT parameter, to which the BFILE file (which must be opened using DBE_LOB.BFILEOPEN first) is loaded. The BFILE file can be greater than 1 GB, up to 32 TB.                                                                                                                                                            |
| src_file    | Source BFILE file to be read. The BFILE file can be greater than 1 GB, up to 32 TB.                                                                                                                                                                                                                                                          |
| amount      | Length of the target BLOB. If the length of a file exceeds this threshold, the file will not be saved to the BLOB.<br><b>NOTE</b><br>If the length is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                                                                                     |
| dest_offset | Offset length of the BLOB.<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                                                                                                                                                                             |
| src_offset  | Offset length of the BFILE file.<br><b>NOTE</b> <ul style="list-style-type: none"> <li>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i>, an error is reported.</li> <li>If the sum of <i>amount</i> and <i>src_offset</i> is greater than the length of <i>src_bfile</i> plus 1, an error is reported.</li> </ul> |

- DBE\_LOB.LOADCLOBFROMFILE

Loads an external BFILE file to a CLOB and returns the object of the RAW type.

The prototype of the DBE\_LOB.LOADCLOBFROMFILE function is as follows:

```
DBE_LOB.LOADCLOBFROMFILE (
 dest_lob IN CLOB,
 src_file IN INTEGER,
 amount IN INTEGER,
 dest_offset IN INTEGER,
 src_offset IN INTEGER)
RETURN RAW;
```

**Table 10-166** DBE\_LOB.LOADCLOBFROMFILE parameters

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_lob    | Target CLOB, to which the BFILE file will be loaded.                                                                                                                                                                                                                                                                                         |
| src_file    | Source BFILE file to be read.                                                                                                                                                                                                                                                                                                                |
| amount      | Length of the CLOB.<br><b>NOTE</b><br>If the length is less than 1 or greater than 32767, an error is reported.                                                                                                                                                                                                                              |
| dest_offset | Offset length of the CLOB.<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                                                                                                                                                                             |
| src_offset  | Offset length of the BFILE file.<br><b>NOTE</b> <ul style="list-style-type: none"> <li>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i>, an error is reported.</li> <li>If the sum of <i>amount</i> and <i>src_offset</i> is greater than the length of <i>src_bfile</i> plus 1, an error is reported.</li> </ul> |

- DBE\_LOB.LOADCLOBFROMBFILE

Loads an external BFILE file to a CLOB.

The prototype of the DBE\_LOB.LOADCLOBFROMBFILE function is as follows:

```
DBE_LOB.LOADCLOBFROMBFILE (
 dest_lob INOUT CLOB,
 src_file IN DEB_LOB.BFILE,
 amount IN BIGINT,
 dest_offset INOUT BIGINT,
 src_offset INOUT BIGINT)
```

**Table 10-167** DBE\_LOB.LOADCLOBFROMBFILE parameters

| Parameter | Description                                                                                                                                                                              |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_lob  | Target CLOB as the INOUT parameter, to which the BFILE file (which must be opened using <b>DBE_LOB.BFILEOPEN</b> first) is loaded. The BFILE file can be greater than 1 GB, up to 32 TB. |
| src_file  | Source BFILE file to be read. The BFILE file can be greater than 1 GB, up to 32 TB.                                                                                                      |

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| amount      | Length of the target CLOB. If the length of a file exceeds this threshold, the file will not be saved to the CLOB.<br><b>NOTE</b><br>If the length is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                                                                                     |
| dest_offset | Offset length of the CLOB.<br><b>NOTE</b><br>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                                                                                                                                                                             |
| src_offset  | Offset length of the BFILE file.<br><b>NOTE</b> <ul style="list-style-type: none"> <li>If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i>, an error is reported.</li> <li>If the sum of <i>amount</i> and <i>src_offset</i> is greater than the length of <i>src_bfile</i> plus 1, an error is reported.</li> </ul> |

- DBE\_LOB.CONVERTTOBLOB

Converts a CLOB to a BLOB. The CLOB cannot be greater than 1 GB.

The prototype of the DBE\_LOB.CONVERTTOBLOB function is as follows:

```
DBE_LOB.CONVERTTOBLOB(
 dest_blob IN BLOB,
 src_clob IN CLOB,
 amount IN INTEGER DEFAULT 32767,
 dest_offset IN INTEGER DEFAULT 1,
 src_offset IN INTEGER DEFAULT 1)
RETURN RAW;
```

**Table 10-168** DBE\_LOB.CONVERTTOBLOB parameters

| Parameter   | Description                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_blob   | Target BLOB, which is converted from a CLOB.                                                                                                                                                                           |
| src_clob    | Source CLOB to be read.                                                                                                                                                                                                |
| amount      | Length of the target CLOB. If the length of a file exceeds this threshold, the file will not be saved to the BLOB. If the length is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported. |
| dest_offset | Offset length of the BLOB. If <b>dest_offset</b> is set to <b>1</b> , data is loaded from the start position of the file. The rest may be deduced by analogy.                                                          |
| src_offset  | Offset length of the CLOB. If <b>src_offset</b> is set to <b>1</b> , data is read from the start position of the file. The rest may be deduced by analogy.                                                             |

- DBE\_LOB.LOB\_CONVERTTOBLOB

Converts a CLOB to a BLOB. The LOB can be greater than 1 GB.

The prototype of the DBE\_LOB.LOB\_CONVERTTOBLOB function is as follows:

```
DBE_LOB.LOB_CONVERTTOBLOB(
 dest_blob INOUT BLOB,
 src_clob IN CLOB,
 amount IN BIGINT,
 dest_offset INOUT BIGINT,
 src_offset INOUT BIGINT)
```

**Table 10-169** DBE\_LOB.LOB\_CONVERTTOBLOB parameters

| Parameter   | Description                                                                                                                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_blob   | Target BLOB, which is converted from a CLOB.                                                                                                                                                                                                                      |
| src_clob    | Source CLOB to be read.                                                                                                                                                                                                                                           |
| amount      | Length of the target CLOB. If the length of a file exceeds this threshold, the file will not be saved to the BLOB. If the length is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.                                            |
| dest_offset | Offset length of the BLOB. If <b>dest_offset</b> is set to <b>1</b> , data is loaded from the start position of the file. The rest may be deduced by analogy. If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported. |
| src_offset  | Offset length of the CLOB. If <b>src_offset</b> is set to <b>1</b> , data is read from the start position of the file. The rest may be deduced by analogy. If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.    |

- DBE\_LOB.CONVERTTOCLOB

Converts a BLOB to a CLOB. The BLOB cannot be greater than 1 GB.

The prototype of the DBE\_LOB.CONVERTTOCLOB function is as follows:

```
DBE_LOB.CONVERTTOCLOB(
 dest_clob IN CLOB,
 src_blob IN BLOB,
 amount IN INTEGER DEFAULT 32767,
 dest_offset IN INTEGER DEFAULT 1,
 src_offset IN INTEGER DEFAULT 1)
RETURN text;
```

**Table 10-170** DBE\_LOB.CONVERTTOCLOB parameters

| Parameter | Description                                                                                                        |
|-----------|--------------------------------------------------------------------------------------------------------------------|
| dest_clob | Target CLOB, which is converted from a BLOB.                                                                       |
| src_blob  | Source BLOB to be read.                                                                                            |
| amount    | Length of the target BLOB. If the length of a file exceeds this threshold, the file will not be saved to the CLOB. |

| Parameter   | Description                                                                                                                                                   |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_offset | Offset length of the CLOB. If <b>dest_offset</b> is set to <b>1</b> , data is loaded from the start position of the file. The rest may be deduced by analogy. |
| src_offset  | Offset length of the BLOB. If <b>src_offset</b> is set to <b>1</b> , data is read from the start position of the file. The rest may be deduced by analogy.    |

- DBE\_LOB.LOB\_CONVERTTOCLOB

Converts a BLOB to a CLOB. The LOB can be greater than 1 GB.

The prototype of the DBE\_LOB.LOB\_CONVERTTOCLOB function is as follows:

```
DBE_LOB.LOB_CONVERTTOCLOB(
 dest_clob INOUT CLOB,
 src_blob IN BLOB,
 amount IN BIGINT,
 dest_offset INOUT BIGINT,
 src_offset INOUT BIGINT)
```

**Table 10-171** DBE\_LOB.LOB\_CONVERTTOCLOB parameters

| Parameter   | Description                                                                                                                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_clob   | Target CLOB, which is converted from a BLOB.                                                                                                                                                                                                                      |
| src_blob    | Source BLOB to be read.                                                                                                                                                                                                                                           |
| amount      | Length of the target BLOB. If the length of a file exceeds this threshold, the file will not be saved to the CLOB.                                                                                                                                                |
| dest_offset | Offset length of the CLOB. If <b>dest_offset</b> is set to <b>1</b> , data is loaded from the start position of the file. The rest may be deduced by analogy. If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported. |
| src_offset  | Offset length of the BLOB. If <b>src_offset</b> is set to <b>1</b> , data is read from the start position of the file. The rest may be deduced by analogy. If the offset is less than 1 or greater than the value of <i>LOBMAXSIZE</i> , an error is reported.    |

- DBE\_LOB.GETCHUNKSIZE

Returns *TOAST\_MAX\_CHUNK\_SIZE*. When LOB data is stored in the database, TOAST is used internally.

The prototype of the DBE\_LOB.GETCHUNKSIZE function is as follows:

```
DBE_LOB.GETCHUNKSIZE(
 lob_loc IN CLOB
)RETURN INTEGER

DBE_LOB.GETCHUNKSIZE(
 lob_loc IN BLOB
)RETURN INTEGER
```

**Table 10-172** DBE\_LOB.GETCHUNKSIZE parameters

| Parameter | Description       |
|-----------|-------------------|
| lob_loc   | Target CLOB/BLOB. |

- DBE\_LOB.LOB\_WRITE

Reads the specified length of the source object from the start position, writes the content to the specified offset position of the target LOB, overrides the original content, and returns the target LOB.

The prototype of the DBE\_LOB.LOB\_WRITE function is as follows:

```
DBE_LOB.LOB_WRITE(
 clob_obj INOUT CLOB,
 amount IN INTEGER,
 off_set IN BIGINT,
 source IN VARCHAR2
)
RETURN CLOB;

DBE_LOB.LOB_WRITE(
 blob_obj INOUT BLOB,
 amount IN INTEGER,
 off_set IN BIGINT,
 source IN RAW
)
RETURN BLOB;
```

**Table 10-173** DBE\_LOB.LOB\_WRITE parameters

| Parameter           | Type           | Input/Output Parameter | Empty or Not | Description                                                                      |
|---------------------|----------------|------------------------|--------------|----------------------------------------------------------------------------------|
| blob_obj / clob_obj | BLOB / CLOB    | INOUT                  | No           | Target LOB as the INOUT parameter, to which the content is to be written.        |
| amount              | INTEGER        | IN                     | No           | Length of the data to be written (in bytes for BLOBs or in characters for CLOBs) |
| off_set             | BIGINT         | IN                     | No           | Offset position for writing data to <i>blob_obj/clob_obj</i>                     |
| source              | RAW / VARCHAR2 | IN                     | No           | Source object                                                                    |

- DBE\_LOB.BFILENAME

Constructs a BFILE based on the directory and file name.

The prototype of DBE\_LOB.BFILENAME is as follows:

```
DBE_LOB.BFILENAME(
 directory IN TEXT,
 filename IN TEXT)
RETURN DBE_LOB.BFILE;
```

**Table 10-174** DBE\_LOB.BFILENAME API parameters

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| directory | File path<br><b>NOTE</b><br>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b> . If the input path does not match the path in <b>PG_DIRECTORY</b> , an error indicating that the path does not exist is reported. <ul style="list-style-type: none"> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to read and write files in the file path specified by <b>safe_data_path</b>.</li> <li>File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the <b>OPEN</b> function. In Unix, the file name cannot end with the combination of a slash and a dot (/).</li> </ul> |
| filename  | File name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## Examples

```
-- Obtain the length of a string.
SELECT DBE_LOB.GET_LENGTH('12345678');
get_length

 8
(1 row)

-- DBE_LOB.READ API
DECLARE
myraw RAW(100);
amount INTEGER :=2;
buffer INTEGER :=1;
begin
DBE_LOB.READ('123456789012345',amount,buffer,myraw);
dbe_output.print_line(myraw);
end;
/
0123
ANONYMOUS BLOCK EXECUTE

CREATE TABLE blob_Table (t1 blob) DISTRIBUTE BY REPLICATION;
CREATE TABLE blob_Table_bak (t2 blob) DISTRIBUTE BY REPLICATION;
INSERT INTO blob_Table VALUES('abcdef');
INSERT INTO blob_Table_bak VALUES('22222');
```

```
-- Multiple DBE_LOB APIs
DECLARE
str varchar2(100) := 'abcdef';
source raw(100);
dest blob;
copyto blob;
amount int;
PSV_SQL varchar2(100);
PSV_SQL1 varchar2(100);
a int :=1;
len int;
```

```

BEGIN
source := dbe_raw.cast_from_varchar2_to_raw(str);
amount := dbe_raw.get_length(source);

PSV_SQL := 'select * from blob_Table for update';
PSV_SQL1 := 'select * from blob_Table_bak for update';

EXECUTE IMMEDIATE PSV_SQL into dest;
EXECUTE IMMEDIATE PSV_SQL1 into copyto;

DBE_LOB.WRITE(dest, amount, 1, source);
DBE_LOB.WRITE_APPEND(dest, amount, source);

DBE_LOB.ERASE(dest, a, 1);
DBE_OUTPUT.PRINT_LINE(a);
DBE_LOB.COPY(copyto, dest, amount, 10, 1);
perform DBE_LOB.CLOSE(dest);
RETURN;
END;
/
1
ANONYMOUS BLOCK EXECUTE

-- Delete the table.
DROP TABLE blob_Table;
DROP TABLE
DROP TABLE blob_Table_bak;
DROP TABLE

```

### 10.11.2.2 DBE\_RANDOM

#### APIs

[Table 10-175](#) lists all APIs supported by the **DBE\_RANDOM** package.

**Table 10-175** DBE\_RANDOM API parameters

| API                                  | Description                                                                            |
|--------------------------------------|----------------------------------------------------------------------------------------|
| <a href="#">DBE_RANDOM.SET_SEED</a>  | Sets a seed for a random number.                                                       |
| <a href="#">DBE_RANDOM.GET_VALUE</a> | Generates a random number between a specified <b>low</b> and a specified <b>high</b> . |

- **DBE\_RANDOM.SET\_SEED**  
The stored procedure SEED is used to set a seed for a random number. The function prototype of DBE\_RANDOM.SET\_SEED is as follows:  
DBE\_RANDOM.SET\_SEED (seed IN INTEGER);

**Table 10-176** DBE\_RANDOM.SET\_SEED API parameters

| Parameter | Description                           |
|-----------|---------------------------------------|
| seed      | Generates a seed for a random number. |

- DBE\_RANDOM.GET\_VALUE

The GET\_VALUE function generates a random number between a specified **low** and a specified **high**. The prototype of the DBE\_RANDOM.GET\_VALUE function is as follows:

```
DBE_RANDOM.GET_VALUE(
min IN NUMBER default 0,
max IN NUMBER default 1)
RETURN NUMBER;
```

**Table 10-177** DBE\_RANDOM.GET\_VALUE API parameters

| Parameter | Description                                                                                                  |
|-----------|--------------------------------------------------------------------------------------------------------------|
| min       | Sets the low bound for a random number. The generated random number is greater than or equal to <b>min</b> . |
| max       | Sets the high bound for a random number. The generated random number is less than <b>max</b> .               |

 **NOTE**

- The only requirement is that the parameter type is **NUMERIC** regardless of the right and left bound values.
- DBE\_RANDOM implements pseudo-random numbers. Therefore, if the initial value (seed) remains unchanged, the sequence of the pseudo-random numbers also remains unchanged.
- The generated random number contains 15 valid digits.

## Examples

```
-- Generate a random number between 0 and 1.
SELECT DBE_RANDOM.GET_VALUE(0,1);
 get_value

.917468812743886
(1 row)
-- For integers within a specified range, add the arguments min and max, and truncate the decimals from
the result (the maximum value is not included as a possible value). Therefore, for integers from 0 to 99, you
can use the following code:
SELECT TRUNC(DBE_RANDOM.GET_VALUE(0,100));
 trunc

 26
(1 row)
```

### 10.11.2.3 DBE\_OUTPUT

 **NOTE**

When DBE\_OUTPUT.PUT\_LINE is used to print the result obtained by the DBE\_FILE.READ\_LINE\_NCHAR API, ensure that the UTF-8 character set encoding can be converted to the current database character set encoding. If the preceding conditions are met, the result can be properly output. DBE\_OUTPUT.PRINT\_LINE does not support this function.

## APIs

**Table DBE\_OUTPUT** provides all APIs supported by the **DBE\_OUTPUT** package.

**Table 10-178** DBE\_OUTPUT

| API                               | Description                                                                                                                                                                                                                        |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DBE_OUTPUT.PRINT_LINE</b>      | Outputs the specified text with newline characters.                                                                                                                                                                                |
| <b>DBE_OUTPUT.PRINT</b>           | Outputs the specified text without newline characters.                                                                                                                                                                             |
| <b>DBE_OUTPUT.SET_BUFFER_SIZE</b> | Sets the size of the output buffer. If the size is not specified, the buffer can contain a maximum of 20000 bytes. If the size is set to a value less than or equal to 2000 bytes, the buffer can contain a maximum of 2000 bytes. |
| <b>DBE_OUTPUT.DISABLE</b>         | Disables the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES, and clears the output buffer.                                                                                                                            |
| <b>DBE_OUTPUT.ENABLE</b>          | Enables the buffer, allows the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES, and sets the buffer size.                                                                                                              |
| <b>DBE_OUTPUT.GET_LINE</b>        | Obtains a line of data from the buffer with a newline character as the boundary. The obtained data is not output to the client.                                                                                                    |
| <b>DBE_OUTPUT.GET_LINES</b>       | Obtains the character string of a specified number of lines in the buffer as a VARCHAR array. The obtained content is cleared from the buffer and is not output to the client.                                                     |
| <b>DBE_OUTPUT.NEW_LINE</b>        | Places a line at the end of the buffer, places an end-of-line marker, and leaves a new line empty.                                                                                                                                 |
| <b>DBE_OUTPUT.PUT</b>             | Places an input string in the buffer without a newline character at the end. When the stored procedure ends, the line ending with the newline character is displayed.                                                              |
| <b>DBE_OUTPUT.PUT_LINE</b>        | Places an input string in the buffer with a newline character at the end. When the stored procedure ends, the line ending with the newline character is displayed.                                                                 |

- **DBE\_OUTPUT.PRINT\_LINE**  
The stored procedure PRINT\_LINE writes a line of text carrying a line end symbol in the buffer. The prototype of the DBE\_OUTPUT.PRINT\_LINE function is as follows:

```
DBE_OUTPUT.PRINT_LINE (
format IN VARCHAR2);
```

**Table 10-179** DBE\_OUTPUT.PRINT\_LINE API parameters

| Parameter | Description  |
|-----------|--------------|
| format    | Output text. |

- **DBE\_OUTPUT.PRINT**  
The stored procedure PRINT outputs the specified text to the front of the specified text without adding a newline character. The prototype of the DBE\_OUTPUT.PRINT function is as follows:

```
DBE_OUTPUT.PRINT (
format IN VARCHAR2);
```

**Table 10-180** DBE\_OUTPUT.PRINT API parameters

| Parameter | Description  |
|-----------|--------------|
| format    | Output text. |

- **DBE\_OUTPUT.SET\_BUFFER\_SIZE**  
The stored procedure SET\_BUFFER\_SIZE sets the output buffer size. If the size is not specified, it contains a maximum of 20000 bytes. The prototype of the DBE\_OUTPUT.SET\_BUFFER\_SIZE function is as follows:

```
DBE_OUTPUT.SET_BUFFER_SIZE (
size IN INTEGER default 20000);
```

**Table 10-181** DBE\_OUTPUT.SET\_BUFFER\_SIZE API parameters

| Parameter | Description                  |
|-----------|------------------------------|
| size      | Sets the output buffer size. |

- **DBE\_OUTPUT.DISABLE**  
The stored procedure DISABLE disables the calling of PUT, PUT\_LINE, NEW\_LINE, GET\_LINE and GET\_LINES, and clears the output buffer. The prototype of the DBE\_OUTPUT.DISABLE function is as follows:

```
DBE_OUTPUT.DISABLE;
```

- **DBE\_OUTPUT.ENABLE**  
The stored procedure ENABLE enables the buffer, allows the calling of PUT, PUT\_LINE, NEW\_LINE, GET\_LINE, and GET\_LINES, and sets the buffer size. If

the buffer size is not specified, 20,000 bytes are allowed by default. The prototype of the DBE\_OUTPUT.ENABLE function is as follows:

```
DBE_OUTPUT.ENABLE (
 buffer_size IN INTEGER DEFAULT 20000
);
```

**Table 10-182** DBE\_OUTPUT.ENABLE parameters

| Parameter   | Description                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------------------|
| buffer_size | Upper limit of the buffer size, in bytes. If <b>buffer_size</b> is set to <b>NULL</b> , the default value (20000 bytes) is used. |

- DBE\_OUTPUT.GET\_LINE

The stored procedure GET\_LINE obtains a line of data from the buffer with a newline character as the boundary. The obtained data is not output to the client. The prototype of the DBE\_OUTPUT.GET\_LINE function is as follows:

```
DBE_OUTPUT.GET_LINE (
 line OUT VARCHAR2,
 status OUT INTEGER
);
```

**Table 10-183** DBE\_OUTPUT.GET\_LINE parameters

| Parameter | Description                                                                                                                            |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------|
| line      | Indicates the obtained character string.                                                                                               |
| status    | Specifies whether the calling is normal. If a character string is obtained, the value is <b>0</b> . Otherwise, the value is <b>1</b> . |

- DBE\_OUTPUT.GET\_LINES

The stored procedure GET\_LINES obtains the character string of a specified number of lines in the buffer as a VARCHAR array. The obtained content is cleared from the buffer and is not output to the client. The prototype of the DBE\_OUTPUT.GET\_LINES function is as follows:

```
DBE_OUTPUT.GET_LINES (
 lines OUT VARCHAR[],
 numlines IN OUT INTEGER
);
```

**Table 10-184** DBE\_OUTPUT.GET\_LINES parameters

| Parameter | Description                                                                                                                                                                                                                                               |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lines     | Outputs an array of multi-line strings read from the buffer.                                                                                                                                                                                              |
| numlines  | Inputs the number of lines to be retrieved from the buffer. The output value is the number of rows actually retrieved. If the output value is less than the input value, it indicates that the number of rows in the buffer is less than the input value. |

- DBE\_OUTPUT.NEW\_LINE

The stored procedure NEW\_LINE places a line at the end of the buffer, places an end-of-line marker, and leaves a new line empty. The prototype of the DBE\_OUTPUT.NEW\_LINE function is as follows:

```
DBE_OUTPUT.NEW_LINE;
```

- DBE\_OUTPUT.PUT

The stored procedure PUT places an input string in the buffer without a newline character at the end. After the anonymous block is executed, the line ending with the newline character is displayed. The prototype of the DBE\_OUTPUT.PUT function is as follows:

```
DBE_OUTPUT.PUT (
 item IN VARCHAR2);
```

**Table 10-185** DBE\_OUTPUT.PUT parameters

| Parameter | Description                                          |
|-----------|------------------------------------------------------|
| item      | Character or string item to be placed in the buffer. |

- DBE\_OUTPUT.PUT\_LINE

The stored procedure PUT\_LINE places an input string in the buffer with a newline character at the end. After the anonymous block is executed, the line ending with the newline character is displayed. The prototype of the DBE\_OUTPUT.PUT\_LINE function is as follows:

```
DBE_OUTPUT.PUT_LINE (
 item IN VARCHAR2);
```

**Table 10-186** DBE\_OUTPUT.PUT\_LINE parameters

| Parameter | Description                                          |
|-----------|------------------------------------------------------|
| item      | Character or string item to be placed in the buffer. |

## Examples

```
BEGIN
 DBE_OUTPUT.SET_BUFFER_SIZE(50);
 DBE_OUTPUT.PRINT('hello, ');
 DBE_OUTPUT.PRINT_LINE('database!');-- Output "hello, database!".
END;
/
-- The expected result is as follows:
hello, database!
ANONYMOUS BLOCK EXECUTE

-- Test DISABLE: Disable the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES. The PUT_LINE
has no output.
BEGIN
 dbe_output.disable();
 dbe_output.put_line('1');
END;
/
-- The expected result is as follows:
ANONYMOUS BLOCK EXECUTE

-- Test ENABLE: Enable the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES. The output of
PUT_LINE is 1.
```

```
BEGIN
 dbe_output.enable();
 dbe_output.put_line('1');
END;
/
-- The expected result is as follows:
1
ANONYMOUS BLOCK EXECUTE

-- Test PUT: Place the input character string a in the buffer without adding a newline character at the end.
a has no output.
BEGIN
 dbe_output.enable();
 dbe_output.put('a');
END;
/
-- The expected result is as follows:
ANONYMOUS BLOCK EXECUTE

-- Test NEW_LINE. Add a new line. The output is a.
BEGIN
 dbe_output.enable();
 dbe_output.put('a');
 dbe_output.new_line;
END;
/
-- The expected result is as follows:
a
ANONYMOUS BLOCK EXECUTE

-- Test GET_LINE: Obtain buffer data and save the data to variables and use PUT_LINE to output the data.
DECLARE
 line VARCHAR(32672);
 status INTEGER := 0;
BEGIN
 dbe_output.put_line('hello');
 dbe_output.get_line(line, status);
 dbe_output.put_line('-----');
 dbe_output.put_line(line);
 dbe_output.put_line(status);
END;
/
The expected result is as follows:

hello
0
ANONYMOUS BLOCK EXECUTE

-- Test GET_LINE: Obtain multiple lines of content in the buffer and use PUT_LINE to output the content.
DECLARE
 lines dbms_output.chararr;
 numlines integer;
BEGIN
 dbe_output.put_line('output line 1');
 dbe_output.put_line('output line 2');
 dbe_output.put_line('output line 3');
 numlines := 100;
 dbe_output.get_lines(lines, numlines);
 dbe_output.put_line('num: ' || numlines);
 dbe_output.put_line('get line 1: ' || lines(1));
 dbe_output.put_line('get line 2: ' || lines(2));
 dbe_output.put_line('get line 3: ' || lines(3));
END;
/
-- The expected result is as follows:
num: 3
get line 1: output line 1
get line 2: output line 2
```

```
get line 3: output line 3
ANONYMOUS BLOCK EXECUTE
```

## 10.11.2.4 DBE\_RAW

### APIs

**Table 10-187** provides all APIs supported by the **DBE\_RAW** package.

**Table 10-187** DBE\_RAW

| API                                                     | Description                                                                  |
|---------------------------------------------------------|------------------------------------------------------------------------------|
| <a href="#">DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW</a> | Converts a value of the INTEGER type to a binary representation (RAW type).  |
| <a href="#">DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER</a> | Converts a binary representation (RAW type) to a value of the INTEGER type.  |
| <a href="#">DBE_RAW.GET_LENGTH</a>                      | Obtains the length of a RAW object.                                          |
| <a href="#">DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW</a>       | Converts a value of the VARCHAR2 type to a binary representation (RAW type). |
| <a href="#">DBE_RAW.CAST_TO_VARCHAR2</a>                | Converts a value of the RAW type to a value of the VARCHAR2 type.            |
| <a href="#">DBE_RAW.SUBSTR</a>                          | Returns the substring of the RAW type.                                       |
| <a href="#">DBE_RAW.BIT_OR</a>                          | Performs the bitwise OR operation on raw data.                               |
| <a href="#">DBE_RAW.BIT_AND</a>                         | Returns a RAW value after bitwise AND calculation.                           |
| <a href="#">DBE_RAW.BIT_COMPLEMENT</a>                  | Returns a RAW value after bitwise complement calculation.                    |
| <a href="#">DBE_RAW.BIT_XOR</a>                         | Returns a RAW value after bitwise XOR calculation.                           |
| <a href="#">DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW</a>  | Converts a BINARY_DOUBLE value to a RAW value.                               |
| <a href="#">DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE</a>  | Converts a RAW value to a BINARY_DOUBLE value.                               |
| <a href="#">DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT</a>   | Converts a RAW value to a FLOAT4 value.                                      |
| <a href="#">DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW</a>   | Converts a FLOAT4 value to a RAW value.                                      |
| <a href="#">DBE_RAW.CAST_FROM_RAW_TO_NUMERIC</a>        | Converts a RAW value to a NUMERIC value.                                     |
| <a href="#">DBE_RAW.CAST_FROM_NUMERIC_TO_RAW</a>        | Converts a NUMERIC value to a RAW value.                                     |

| API                                                | Description                                                                                                            |
|----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2</a> | Converts a RAW value to an NVARCHAR2 value.                                                                            |
| <a href="#">DBE_RAW.COMPARE</a>                    | Returns the first different position of two RAW values.                                                                |
| <a href="#">DBE_RAW.CONCAT</a>                     | Concatenates a maximum of 12 RAW values into a new RAW value and returns the value.                                    |
| <a href="#">DBE_RAW.CONVERT</a>                    | Converts a RAW value from the source encoding mode <b>from_charset</b> to the target encoding mode <b>to_charset</b> . |
| <a href="#">DBE_RAW.COPIES</a>                     | Copies a RAW value for <i>n</i> times, concatenates the values, and returns the concatenated result.                   |
| <a href="#">DBE_RAW.OVERLAY</a>                    | Overlays one RAW data with another RAW data by specifying the start position and length to be overlaid.                |
| <a href="#">DBE_RAW.REVERSE</a>                    | Reverses RAW data by byte.                                                                                             |
| <a href="#">DBE_RAW.TRANSLATE</a>                  | Converts or discards a specified byte of a RAW value.                                                                  |
| <a href="#">DBE_RAW.TRANSLITERATE</a>              | Converts a specified byte of a RAW value.                                                                              |
| <a href="#">DBE_RAW.XRANGE</a>                     | Returns a RAW value containing the succession of one-byte encodings beginning and ending with the specified byte-code. |

#### NOTICE

RAW data is represented as hexadecimal characters externally, and stored as binary characters internally. For example, the representation of RAW data 11001011 is 'CB', that is, the input for type conversion is 'CB'.

- [DBE\\_RAW.CAST\\_FROM\\_BINARY\\_INTEGER\\_TO\\_RAW](#)  
The stored procedure `CAST_FROM_BINARY_INTEGER_TO_RAW` converts a value of the `INTEGER` type to a binary representation (RAW type).

The prototype of the `DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW` function is as follows:

```
DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW (
value IN INTEGER,
endianess IN INTEGER DEFAULT 1)
RETURN RAW;
```

**Table 10-188** DBE\_RAW.CAST\_FROM\_BINARY\_INTEGER\_TO\_RAW parameters

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| value      | Specifies the INTEGER value to be converted to the RAW value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| endianness | Endianness. The value can be <b>1</b> , <b>2</b> , or <b>3</b> . The value <b>1</b> indicates BIG_ENDIAN, <b>2</b> indicates LITTLE_ENDIAN, and <b>3</b> indicates MACHINE_ENDIAN. The default value is <b>1</b> . If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN. |

- DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_INTEGER

The stored procedure CAST\_FROM\_RAW\_TO\_BINARY\_INTEGER converts a binary representation (RAW type) to a value of the INTEGER type.

The prototype of the DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_INTEGER function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER (
value IN RAW,
endianness IN INTEGER DEFAULT 1)
RETURN INTEGER;
```

**Table 10-189** DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_INTEGER parameters

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| value      | Specifies an INTEGER value in a binary representation (RAW type).                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| endianness | Endianness. The value can be <b>1</b> , <b>2</b> , or <b>3</b> . The value <b>1</b> indicates BIG_ENDIAN, <b>2</b> indicates LITTLE_ENDIAN, and <b>3</b> indicates MACHINE_ENDIAN. The default value is <b>1</b> . If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN. |

- DBE\_RAW.GET\_LENGTH

The stored procedure GET\_LENGTH returns the length of a RAW object.

The prototype of the DBE\_RAW.GET\_LENGTH function is as follows:

```
DBE_RAW.GET_LENGTH(
value IN RAW)
RETURN INTEGER;
```

**Table 10-190** DBE\_RAW.GET\_LENGTH parameters

| Parameter | Description             |
|-----------|-------------------------|
| value     | Specifies a RAW object. |

- DBE\_RAW.CAST\_FROM\_VARCHAR2\_TO\_RAW

The stored procedure CAST\_FROM\_VARCHAR2\_TO\_RAW converts a VARCHAR2 object to a RAW object.

The prototype of the DBE\_RAW.CAST\_FROM\_VARCHAR2\_TO\_RAW function is as follows:

```
DBE_RAW.CAST_TO_RAW(
str IN VARCHAR2)
RETURN RAW;
```

**Table 10-191** DBE\_RAW.CAST\_FROM\_VARCHAR2\_TO\_RAW parameters

| Parameter | Description                     |
|-----------|---------------------------------|
| str       | VARCHAR2 value to be converted. |

- DBE\_RAW.CAST\_TO\_VARCHAR2

The stored procedure CAST\_TO\_VARCHAR2 converts a RAW object to a VARCHAR2 object.

The prototype of the DBE\_RAW.CAST\_TO\_VARCHAR2 function is as follows:

```
DBE_RAW.CAST_TO_VARCHAR2(
str IN RAW)
RETURN VARCHAR2;
```

**Table 10-192** DBE\_RAW.CAST\_TO\_VARCHAR2 parameters

| Parameter | Description                                    |
|-----------|------------------------------------------------|
| str       | Specifies a <b>RAW</b> object to be converted. |

- DBE\_RAW.BIT\_OR

The stored procedure BIT\_OR calculates the bitwise OR result of two raw data records.

The prototype of the DBE\_RAW.BIT\_OR function is as follows:

```
DBE_RAW.BIT_OR(
str1 IN TEXT,
str2 IN TEXT)
RETURN TEXT;
```

**Table 10-193** DBE\_RAW.BIT\_OR parameters

| Parameter | Description                                                                                                                                                                                                                                                                                       |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| str1      | First character string of the bitwise OR calculation.<br><b>NOTE</b><br>Due to legacy reasons, this parameter is defined as the TEXT type. However, the DBE_RAW.SUBSTR API expects to process RAW values. Defining this parameter as the TEXT type does not affect the processing of RAW values.  |
| str2      | Second character string of the bitwise OR calculation.<br><b>NOTE</b><br>Due to legacy reasons, this parameter is defined as the TEXT type. However, the DBE_RAW.SUBSTR API expects to process RAW values. Defining this parameter as the TEXT type does not affect the processing of RAW values. |

- DBE\_RAW.SUBSTR

The stored procedure SUBSTR truncates an object of the RAW type based on the start bit and length.

The prototype of the DBE\_RAW.SUBSTR function is as follows:

```
DBE_RAW.SUBSTR(
 IN lob_loc BLOB,
 IN off_set integer default 1,
 IN amount integer default 32767)
RETURN RAW;
```

**Table 10-194** DBE\_RAW.SUBSTR parameters

| Parameter | Description                                                                                                                                                                                                                                                  |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lob_loc   | Source RAW value.<br><b>NOTE</b><br>Due to legacy reasons, this parameter is defined as the BLOB type. However, the DBE_RAW.SUBSTR API expects to process RAW values. Defining this parameter as the BLOB type does not affect the processing of RAW values. |
| off_set   | Start position of the substring. The default value is <b>1</b> .                                                                                                                                                                                             |
| amount    | Substring length. The default value is <b>32767</b> .                                                                                                                                                                                                        |

- DBE\_RAW.BIT\_AND

Obtains the bitwise AND result of two RAW values.

The prototype of the DBE\_RAW.BIT\_AND function is as follows:

```
DBE_RAW.BIT_AND(
 r1 IN RAW,
 r2 IN RAW)
RETURN RAW;
```

**Table 10-195** DBE\_RAW.BIT\_AND parameters

| Parameter | Description                                                                       |
|-----------|-----------------------------------------------------------------------------------|
| r1        | The RAW value for the bitwise AND operation with r2. The maximum length is 32767. |
| r2        | The RAW value for the bitwise AND operation with r1. The maximum length is 32767. |

- DBE\_RAW.BIT\_COMPLEMENT

Obtains the bitwise complement result of a RAW value.

The prototype of the DBE\_RAW.BIT\_COMPLEMENT function is as follows:

```
DBE_RAW.BIT_COMPLEMENT(
 r IN RAW)
RETURN RAW;
```

**Table 10-196** DBE\_RAW.BIT\_COMPLEMENT parameters

| Parameter | Description                                                                      |
|-----------|----------------------------------------------------------------------------------|
| r         | The RAW value for the bitwise complement operation. The maximum length is 32767. |

- DBE\_RAW.BIT\_XOR

Obtains the bitwise XOR result of two RAW values.

The prototype of the DBE\_RAW.BIT\_XOR function is as follows:

```
DBE_RAW.BIT_XOR(
 r1 IN RAW,
 r2 IN RAW)
RETURN RAW;
```

**Table 10-197** DBE\_RAW.BIT\_XOR parameters

| Parameter | Description                                                                       |
|-----------|-----------------------------------------------------------------------------------|
| r1        | The RAW value for the bitwise XOR operation with r2. The maximum length is 32767. |
| r2        | The RAW value for the bitwise XOR operation with r1. The maximum length is 32767. |

- DBE\_RAW.CAST\_FROM\_BINARY\_DOUBLE\_TO\_RAW

Converts the BINARY\_DOUBLE type to an INTEGER value in a binary representation (RAW type).

The prototype of the DBE\_RAW.CAST\_FROM\_BINARY\_DOUBLE\_TO\_RAW function is as follows:

```
DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW (
 n IN BINARY_DOUBLE,
 endianness IN INTEGER DEFAULT 1)
RETURN RAW;
```

**Table 10-198** DBE\_RAW.CAST\_FROM\_BINARY\_DOUBLE\_TO\_RAW parameters

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n          | BINARY_DOUBLE value to be converted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| endianness | Endianness. The value can be <b>1</b> , <b>2</b> , or <b>3</b> . The value <b>1</b> indicates BIG_ENDIAN, <b>2</b> indicates LITTLE_ENDIAN, and <b>3</b> indicates MACHINE_ENDIAN. The default value is <b>1</b> . If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN. |

- DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_DOUBLE

Converts an INTEGER value in a binary representation (RAW type) to a BINARY\_DOUBLE type.

The prototype of the DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_DOUBLE function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE(
 r IN RAW,
 endianness IN INTEGER DEFAULT 1)
RETURN BINARY_DOUBLE;
```

**Table 10-199** DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_DOUBLE parameters

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| r          | The RAW value to be converted. The value contains 8 to 32767 characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| endianness | Endianness. The value can be <b>1</b> , <b>2</b> , or <b>3</b> . The value <b>1</b> indicates BIG_ENDIAN, <b>2</b> indicates LITTLE_ENDIAN, and <b>3</b> indicates MACHINE_ENDIAN. The default value is <b>1</b> . If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN. |

- DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_FLOAT

Converts a RAW value to a FLOAT4 value.

The prototype of the DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_FLOAT function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT(
 r IN RAW,
 endianness IN INTEGER DEFAULT 1)
RETURN FLOAT4;
```

**Table 10-200** DBE\_RAW.CAST\_FROM\_RAW\_TO\_BINARY\_FLOAT parameters

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| r         | The RAW value to be converted. The value contains 4 to 32767 characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| endianess | Endianness. The value can be <b>1</b> , <b>2</b> , or <b>3</b> . The value <b>1</b> indicates BIG_ENDIAN, <b>2</b> indicates LITTLE_ENDIAN, and <b>3</b> indicates MACHINE_ENDIAN. The default value is <b>1</b> . If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN. |

- DBE\_RAW.CAST\_FROM\_BINARY\_FLOAT\_TO\_RAW

Converts a FLOAT4 value to a RAW value.

The prototype of the DBE\_RAW.CAST\_FROM\_BINARY\_FLOAT\_TO\_RAW function is as follows:

```
DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW(
n IN FLOAT4,
endianess IN INTEGER DEFAULT 1)
RETURN RAW;
```

**Table 10-201** DBE\_RAW.CAST\_FROM\_BINARY\_FLOAT\_TO\_RAW parameters

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n         | FLOAT4 value to be converted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| endianess | Endianness. The value can be <b>1</b> , <b>2</b> , or <b>3</b> . The value <b>1</b> indicates BIG_ENDIAN, <b>2</b> indicates LITTLE_ENDIAN, and <b>3</b> indicates MACHINE_ENDIAN. The default value is <b>1</b> . If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN. |

- DBE\_RAW.CAST\_FROM\_RAW\_TO\_NUMBER

Converts a RAW value to a NUMERIC value.

 **NOTE**

The bottom-layer implementation of the number data type in database ORA is different from that in GaussDB, and the raw data type is the hexadecimal representation of the binary stream implemented at the bottom layer. Therefore, the function in database ORA is different from that in GaussDB. You cannot obtain the same number-type data from GaussDB based on the raw data corresponding to the number-type data in database ORA. For details about the number-type data in GaussDB, see the example.

The prototype of the DBE\_RAW.CAST\_FROM\_RAW\_TO\_NUMBER function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_NUMBER(
 r IN RAW)
RETURN NUMERIC;
```

**Table 10-202** DBE\_RAW.CAST\_FROM\_RAW\_TO\_NUMBER parameters

| Parameter | Description                                                              |
|-----------|--------------------------------------------------------------------------|
| r         | The RAW value to be converted. The value contains 6 to 32767 characters. |

- DBE\_RAW.CAST\_FROM\_NUMBER\_TO\_RAW  
Converts a NUMERIC value to a RAW value.

 **NOTE**

The bottom-layer implementation of the number data type in database ORA is different from that in GaussDB, and the raw data type is the hexadecimal representation of the binary stream implemented at the bottom layer. Therefore, the function in database ORA is different from that in GaussDB. You cannot restore the raw data corresponding to the number-type data in database ORA to the original number type in database ORA in the GaussDB database. For details about the performance of this function in GaussDB, see the example.

The prototype of the DBE\_RAW.CAST\_FROM\_NUMBER\_TO\_RAW function is as follows:

```
DBE_RAW.CAST_FROM_NUMBER_TO_RAW(
 n IN NUMERIC)
RETURN RAW;
```

**Table 10-203** DBE\_RAW.CAST\_FROM\_NUMBER\_TO\_RAW parameters

| Parameter | Description                    |
|-----------|--------------------------------|
| n         | NUMERIC value to be converted. |

- DBE\_RAW.CAST\_FROM\_RAW\_TO\_NVARCHAR2  
Converts a RAW value to an NVARCHAR2 value.

The prototype of the DBE\_RAW.CAST\_FROM\_RAW\_TO\_NVARCHAR2 function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2(
 r IN RAW)
RETURN NVARCHAR2;
```

**Table 10-204** DBE\_RAW.CAST\_FROM\_RAW\_TO\_NVARCHAR2 parameters

| Parameter | Description                                                 |
|-----------|-------------------------------------------------------------|
| r         | The RAW value to be converted. The maximum length is 32767. |

- DBE\_RAW.COMPARE

Returns the first different position of two RAW values.

The prototype of the DBE\_RAW.COMPARE function is as follows:

```
DBE_RAW.COMPARE(
 r1 IN RAW,
 r2 IN RAW,
 pad IN RAW DEFAULT NULL)
RETURN INTEGER;
```

**Table 10-205** DBE\_RAW.COMPARE parameters

| Parameter | Description                                                                                                                                                                                                                                          |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| r1        | First data to be compared. The value can be <b>NULL</b> or the length is 0. The maximum length is 32767.                                                                                                                                             |
| r2        | Second data to be compared. The value can be <b>NULL</b> or the length is 0. The maximum length is 32767.                                                                                                                                            |
| pad       | The first byte of <b>pad</b> is used to extend the shorter one of <b>r1</b> or <b>r2</b> . The maximum length is 32767. If this parameter is set to <b>NULL</b> , the length is 0, or the default value is used, the extended value is <b>0x00</b> . |

- DBE\_RAW.CONCAT

Concatenates a maximum of 12 RAW values into a new RAW value and returns the value. If the length after concatenation exceeds 32767, an error is reported.

The prototype of the DBE\_RAW.CONCAT function is as follows:

```
DBE_RAW.CONCAT(
 r1 IN RAW DEFAULT NULL,
 r2 IN RAW DEFAULT NULL,
 r3 IN RAW DEFAULT NULL,
 r4 IN RAW DEFAULT NULL,
 r5 IN RAW DEFAULT NULL,
 r6 IN RAW DEFAULT NULL,
 r7 IN RAW DEFAULT NULL,
 r8 IN RAW DEFAULT NULL,
 r9 IN RAW DEFAULT NULL,
 r10 IN RAW DEFAULT NULL,
 r11 IN RAW DEFAULT NULL,
 r12 IN RAW DEFAULT NULL)
RETURN RAW;
```

**Table 10-206** DBE\_RAW.CONCAT parameters

| Parameter | Description                        |
|-----------|------------------------------------|
| r1...r12  | The RAW values to be concatenated. |

- DBE\_RAW.CONVERT

Converts a RAW value from the source encoding mode **from\_charset** to the target encoding mode **to\_charset**.

If the rule for converting between source and target encoding (for example, GBK and LATIN1) does not exist, the parameter **r** is returned without conversion. See the pg\_conversion system catalog for details.

The prototype of the DBE\_RAW.CONVERT function is as follows:

```
DBE_RAW.CONVERT(
 r IN RAW,
 to_charset IN VARCHAR2,
 from_charset IN VARCHAR2)
RETURN RAW;
```

**Table 10-207** DBE\_RAW.CONVERT parameters

| Parameter    | Description                                                          |
|--------------|----------------------------------------------------------------------|
| r            | The RAW value to be converted. The maximum length is 32767.          |
| to_charset   | Name of the target character set.                                    |
| from_charset | Name of the source character set. In this encoding, r must be valid. |

- DBE\_RAW.COPIES

Copies a RAW value for *n* times, concatenates the values, and returns the concatenated result. If the length after copying exceeds 32767, an error is reported.

The prototype of the DBE\_RAW.COPIES function is as follows:

```
DBE_RAW.COPIES(
 r IN RAW,
 n IN NUMERIC)
RETURN RAW;
```

**Table 10-208** DBE\_RAW.COPIES parameters

| Parameter | Description                                                |
|-----------|------------------------------------------------------------|
| r         | The RAW values to be copied.                               |
| n         | Number of copy times. The value must be a positive number. |

- DBE\_RAW.OVERLAY

Overlays one RAW data with another RAW data by specifying the start position and length to be overlaid.

The prototype of the DBE\_RAW.OVERLAY function is as follows:

```
DBE_RAW.OVERLAY(
 overlay_str IN RAW,
 target IN RAW,
 pos IN BINARY_INTEGER DEFAULT 1,
 len IN BINARY_INTEGER DEFAULT NULL,
 pad IN RAW DEFAULT NULL)
RETURN RAW;
```

**Table 10-209** DBE\_RAW.OVERLAY parameters

| Parameter   | Description                                                  |
|-------------|--------------------------------------------------------------|
| overlay_str | Byte used for overwriting. The value cannot be <b>NULL</b> . |

| Parameter | Description                                                                                                                                                                                                                                            |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| target    | Source byte string to be overlaid. The value is of the RAW type and contains a maximum of 32767 bytes. The value cannot be <b>NULL</b> .                                                                                                               |
| pos       | Indicates the byte from which the overlay starts. The position of the first byte is 1. The value of <b>pos</b> must be greater than or equal to 1 and the value of <b>len+pos</b> must be less than or equal to 32767. The default value is <b>1</b> . |
| len       | Length to be overlaid. The value of <b>len</b> must be greater than or equal to 0 and the value of <b>len+pos</b> must be less than or equal to 32767. The default value is the length of <b>overlay_str</b> .                                         |
| pad       | Padding byte. Only the first byte is valid. The default value is <b>NULL</b> . If the value is <b>NULL</b> , it is regarded as <b>0x00</b> by default.                                                                                                 |

- DBE\_RAW.REVERSE

Reverses RAW data by byte.

The prototype of the DBE\_RAW.REVERSE function is as follows:

```
DBE_RAW.REVERSE(
 r IN RAW
)
RETURN RAW;
```

**Table 10-210** DBE\_RAW.REVERSE parameters

| Parameter | Description                                                                                                       |
|-----------|-------------------------------------------------------------------------------------------------------------------|
| r         | The RAW value to be reversed. The maximum length is 32767. If the value is <b>NULL</b> , <b>NULL</b> is returned. |

- DBE\_RAW.TRANSLATE

Converts or discards a specified byte of a RAW value.

The prototype of the DBE\_RAW.TRANSLATE function is as follows:

```
DBE_RAW.TRANSLATE(
 r IN RAW,
 from_set IN RAW,
 to_set IN RAW)
RETURN RAW;
```

**Table 10-211** DBE\_RAW.TRANSLATE parameters

| Parameter | Description                                                                                                                               |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------|
| r         | Source byte string to be converted. The value is of the RAW type and contains a maximum of 32767 bytes. The value cannot be <b>NULL</b> . |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| from_set  | Bytecode to be converted in the source byte string. The value is of the RAW type. The value cannot be <b>NULL</b> . The bytes in <b>from_set</b> in the source byte string are replaced with the bytes in the corresponding positions in <b>to_set</b> . If <b>from_set</b> contains multiple identical bytes, only the first byte corresponding to the byte is replaced. For example, if <b>r[x]=from_set[n]</b> , <b>r[x]</b> is replaced with <b>to_set[n]</b> . If <b>to_set[n]</b> corresponding to <b>from_set[n]</b> does not exist (that is, the number of bytes of <b>to_set</b> does not exceed <b>n</b> ), <b>r[x]</b> will be discarded. |
| to_set    | Byte code converted from the <b>from_set</b> byte. The value is of the RAW type. The value cannot be <b>NULL</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

- DBE\_RAW.TRANSLITERATE

Converts a RAW value to a NUMERIC value.

The prototype of the DBE\_RAW.TRANSLITERATE function is as follows:

```
DBE_RAW.TRANSLITERATE(
 r IN RAW,
 from_set IN RAW DEFAULT NULL,
 to_set IN RAW DEFAULT NULL,
 pad IN RAW DEFAULT NULL)
RETURN RAW;
```

**Table 10-212** DBE\_RAW.TRANSLITERATE parameters

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| r         | Source byte string to be converted. The value is of the RAW type and contains a maximum of 32767 bytes. The value cannot be <b>NULL</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| to_set    | Byte code converted from the <b>from_set</b> byte. The value is of the RAW type. The default value is <b>NULL</b> . If the value is <b>NULL</b> , all bytes in <b>r</b> that exist in <b>from_set</b> are converted to <b>pad</b> .                                                                                                                                                                                                                                                                                                                                                                            |
| from_set  | Bytecode to be converted in the source byte string <b>r</b> . The value is of the RAW type. The default value is <b>NULL</b> . If <b>from_set</b> is <b>NULL</b> , all bytes in the source byte string <b>r</b> are converted into equal-length data filled by <b>pad</b> . Otherwise, the bytes in <b>from_set</b> in the source byte string <b>r</b> are replaced with the bytes in the corresponding position in <b>to_set</b> . For example, if <b>r[x]=from_set[n]</b> , <b>r[x]</b> is converted to <b>to_set[n]</b> . If <b>to_set[n]</b> does not exist, <b>r[x]</b> will be converted to <b>pad</b> . |
| pad       | Padding byte. Only the first byte is valid. The default value is <b>NULL</b> . If the value is <b>NULL</b> , it is regarded as <b>0x00</b> by default.                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

- DBE\_RAW.XRANGE

Returns a RAW value containing the succession of one-byte encodings beginning and ending with the specified byte-code.

The prototype of the DBE\_RAW.XRANGE function is as follows:

```
DBE_RAW.XRANGE(
 start_byte IN RAW,
 end_byte IN RAW)
RETURN RAW;
```

**Table 10-213** DBE\_RAW.XRANGE parameters

| Parameter  | Description                                                                                                                                                                                                                                                                      |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| start_byte | Start byte. Only the first byte is valid. If the value is <b>NULL</b> , it is regarded as <b>0x00</b> by default.                                                                                                                                                                |
| end_byte   | End byte. Only the first byte is valid. If the value is <b>NULL</b> , it is regarded as <b>0xFF</b> by default. If <b>end_byte</b> is less than <b>start_byte</b> , <b>end_byte</b> is concatenated to <b>0xFF</b> , and then <b>0x00</b> is concatenated to <b>start_byte</b> . |

## Examples

```
--Perform operations on RAW data in a stored procedure.
CREATE OR REPLACE PROCEDURE proc_raw
AS
str varchar2(100) := 'abcdef';
source raw(100);
amount integer;
BEGIN
source := dbe_raw.cast_from_varchar2_to_raw(str);--Convert the type.
amount := dbe_raw.get_length(source);--Obtain the length.
dbe_output.print_line(amount);
END;
/
CREATE PROCEDURE

-- Call the stored procedure.
CALL proc_raw();
6
proc_raw

(1 row)
-- Drop the stored procedure.
DROP PROCEDURE proc_raw;
DROP PROCEDURE

DECLARE
v_raw RAW;
v_double BINARY_DOUBLE;
v_float FLOAT4;
v_numeric NUMERIC;
v_nvarchar2 NVARCHAR2;
BEGIN
-- Perform bitwise AND calculation on RAW values.
SELECT DBE_RAW.BIT_AND('AFF', 'FF0B') INTO v_raw; -- 0A0B
-- Perform bitwise complement calculation on RAW values.
SELECT DBE_RAW.BIT_COMPLEMENT('0AFF') INTO v_raw; -- F500
-- Perform bitwise XOR calculation on RAW values.
SELECT DBE_RAW.BIT_XOR('AFF', 'FF0B') INTO v_raw; -- F5F4
-- Convert a BINARY_DOUBLE value to a RAW value.
SELECT DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW(1.0001,1) INTO v_raw; -- 3FF00068DB8BAC71
-- Convert a RAW value to a BINARY_DOUBLE value.
SELECT DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE('3FF00068DB8BAC71',1) INTO v_double; -- 1.0001
-- Convert a RAW value to a FLOAT4 value.
SELECT DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT('40200000',1) INTO v_float; -- 2.5
```

```
-- Convert a FLOAT4 value to a RAW value.
SELECT DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW('2.5',1) INTO v_raw; -- 40200000
-- Convert a RAW value to a NUMERIC value.
SELECT DBE_RAW.CAST_FROM_RAW_TO_NUMBER('808002008813') INTO v_numeric; -- 2.5
-- Convert a NUMERIC value to a RAW value.
SELECT DBE_RAW.CAST_FROM_NUMBER_TO_RAW('2.5') INTO v_raw; -- 808002008813
-- Convert a RAW value to an NVARCHAR2 value.
SELECT DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2('12345678') INTO v_nvarchar2; -- \x124Vx
-- Compare RAW values.
SELECT DBE_RAW.COMPARE('ABCD','AB') INTO v_numeric; -- 2
-- Concatenate RAW values.
SELECT DBE_RAW.CONCAT('ABCD','AB') INTO v_raw; -- ABCDAB
-- Convert RAW values.
SELECT DBE_RAW.CONVERT('E695B0', 'GBK','UTF8') INTO v_raw; -- CAFD
-- Copy RAW values.
SELECT DBE_RAW.COPIES('ABCD',2) INTO v_raw; -- ABCDABCD
-- Specify the start position and length of a RAW value to be overlaid.
SELECT DBE_RAW.OVERLAY('abcef', '12345678123456', 2, 5, '9966') INTO v_raw; -- 120ABCEF999956
-- Reverse a RAW value by byte.
SELECT DBE_RAW.REVERSE('12345678') INTO v_raw; -- 78563412
-- Convert bytes of the RAW type (without padding code)
SELECT DBE_RAW.TRANSLATE('1122112233', '1133','55') INTO v_raw; -- 55225522
-- Convert bytes of the RAW type (with padding code)
SELECT DBE_RAW.TRANSLITERATE('1122112233', '55','1133','FFEE') INTO v_raw; -- 55225522FF
-- All bytes between two bytes of the RAW type.
SELECT DBE_RAW.XRANGE('00','03') INTO v_raw; -- 00010203
END;
/
ANONYMOUS BLOCK EXECUTE
```

### 10.11.2.5 DBE\_TASK

#### APIs

**Table DBE\_TASK** lists all APIs supported by the **DBE\_TASK** package.

**Table 10-214 DBE\_TASK**

| API                        | Description                                                                                                                      |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>DBE_TASK.SUBMIT</b>     | Commits a scheduled job. The job ID is automatically generated by the system.                                                    |
| <b>DBE_TASK.JOB_SUBMIT</b> | Same as <b>DBE_TASK.SUBMIT</b> . However, It provides syntax compatibility parameters.                                           |
| <b>DBE_TASK.ID_SUBMIT</b>  | Commits a scheduled job. The job ID is specified by the user.                                                                    |
| <b>DBE_TASK.CANCEL</b>     | Removes a scheduled task by job ID.                                                                                              |
| <b>DBE_TASK.RUN</b>        | Executes a scheduled task.                                                                                                       |
| <b>DBE_TASK.FINISH</b>     | Disables or enables scheduled task execution.                                                                                    |
| <b>DBE_TASK.UPDATE</b>     | Modifies user-definable attributes of a scheduled task, including the task content, next-execution time, and execution interval. |
| <b>DBE_TASK.CHANGE</b>     | Same as <b>DBE_TASK.UPDATE</b> . However, It provides syntax compatibility parameters.                                           |

| API                                | Description                                                     |
|------------------------------------|-----------------------------------------------------------------|
| <a href="#">DBE_TASK.CONTENT</a>   | Modifies the content attribute of a scheduled task.             |
| <a href="#">DBE_TASK.NEXT_TIME</a> | Modifies the next-execution time attribute of a scheduled task. |
| <a href="#">DBE_TASK.INTERVAL</a>  | Modifies the execution interval attribute of a scheduled task.  |

- DBE\_TASK.SUBMIT

The stored procedure SUBMIT submits a scheduled task provided by the system.

The prototype of the DBE\_TASK.SUBMIT function is as follows:

```
DBE_TASK.SUBMIT(
what IN TEXT,
next_time IN TIMESTAMP DEFAULT sysdate,
interval_time IN TEXT DEFAULT 'null',
id OUT INTEGER
)RETURN INTEGER;
```

 **NOTE**

When a scheduled task is created (using **DBE\_TASK**), the system binds the current database and the username to the task by default. The function can be called using CALL or SELECT. If the function is called using SELECT, you do not need to set the output parameter. If the function is called using CALL, you need to set the output parameter. To call this function within a stored procedure, use **perform**. If the committed SQL statement task uses a non-public schema, specify the schema to a table schema or a function schema, or add **set current\_schema = xxx** before the SQL statement.

**Table 10-215** DBE\_TASK.SUBMIT parameters

| Parameter | Type      | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                               |
|-----------|-----------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| what      | Text      | IN                     | No           | SQL statement to be executed. One or multiple DDL operations (excluding database-related operations), DML operations, anonymous blocks, and statements for calling stored procedures, or all four combined are supported. |
| next_time | Timestamp | IN                     | No           | Specifies the next time the job will be executed. The default value is the current system time ( <b>sysdate</b> ). If the specified time has past, the job is executed at the time it is committed.                       |

| Parameter     | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                     |
|---------------|---------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| interval_time | Text    | IN                     | Yes          | Calculates the next time to execute the job. It can be an interval expression, or <b>sysdate</b> followed by a numeric value, for example, <b>sysdate+1.0/24</b> . If this parameter is left empty or set to <b>null</b> , the job will be executed only once, and the job status will change to 'd' afterward. |
| id            | Integer | OUT                    | No           | Specifies the job ID. The value ranges from 1 to 32767. When SELECT is used for calling, this parameter cannot be added. When CALL is used for calling, this parameter must be added.                                                                                                                           |

**NOTICE**

When you create a user using the **what** parameter, the plaintext password of the user is recorded in the log. Therefore, you are advised not to do so. Tasks created using this API may not be highly available. You are advised to use PKG\_SERVICE.SUBMIT\_ON\_NODES to create a task and specify the CCN as the job execution node.

Example:

```
gaussdb=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1');
submit

 31031
(1 row)

gaussdb=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
submit

 512
(1 row)

gaussdb=# DECLARE
gaussdb-# jobid int;
gaussdb-# BEGIN
gaussdb$$# PERFORM DBE_TASK.SUBMIT('call pro_xxx();', sysdate, 'interval "5 minute"', jobid);
gaussdb$$# END;
gaussdb$$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE\_TASK.JOB\_SUBMIT

The stored procedure SUBMIT commits a scheduled job provided by the system. In addition, it provides additional compatibility parameters.

The prototype of DBE\_TASK.JOB\_SUBMIT is as follows:

```
DBE_TASK.JOB_SUBMIT(
job OUT INTEGER,
```

```

what IN TEXT,
next_date IN TIMESTAMP DEFAULT sysdate,
job_interval IN TEXT DEFAULT 'null',
no_parse IN BOOLEAN DEFAULT false,
instance IN INTEGER DEFAULT 0,
force IN BOOLEAN DEFAULT false
)RETURN INTEGER;

```

**Table 10-216** DBE\_TASK.JOB\_SUBMIT parameters

| Parameter    | Type      | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                     |
|--------------|-----------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| job          | Integer   | OUT                    | No           | Specifies the job ID. The value ranges from 1 to 32767. When dbe.job_submit is called using SELECT, this parameter can be omitted.                                                                                                                                                                              |
| what         | Text      | IN                     | No           | SQL statement to be executed. One or multiple DDL operations (excluding database-related operations), DML operations, anonymous blocks, and statements for calling stored procedures, or all four combined are supported.                                                                                       |
| next_date    | Timestamp | IN                     | Yes          | Specifies the next time the job will be executed. The default value is the current system time ( <b>sysdate</b> ). If the specified time has past, the job is executed at the time it is committed.                                                                                                             |
| job_interval | Text      | IN                     | Yes          | Calculates the next time to execute the job. It can be an interval expression, or <b>sysdate</b> followed by a numeric value, for example, <b>sysdate+1.0/24</b> . If this parameter is left empty or set to <b>null</b> , the job will be executed only once, and the job status will change to 'd' afterward. |
| no_parse     | Boolean   | IN                     | Yes          | The default value is <b>false</b> , which is used only for syntax compatibility.                                                                                                                                                                                                                                |
| instance     | Integer   | IN                     | Yes          | The default value is <b>0</b> , which is used only for syntax compatibility.                                                                                                                                                                                                                                    |
| force        | Boolean   | IN                     | Yes          | The default value is <b>false</b> , which is used only for syntax compatibility.                                                                                                                                                                                                                                |

Example:

```

gaussdb=# DECLARE
gaussdb=# id integer;
gaussdb=# BEGIN

```

```
gaussdb$# id = DBE_TASK.JOB_SUBMIT(
gaussdb$# what => 'insert into t1 values (1, 2)',
gaussdb$# job_interval => 'sysdate + 1' --daily
gaussdb$#);
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- **DBE\_TASK.ID\_SUBMIT**

ID\_SUBMIT has the same syntax function as SUBMIT, but the first parameter of ID\_SUBMIT is an input parameter, that is, a specified job ID. In contrast, that last parameter of ID\_SUBMIT is an output parameter, indicating the job ID automatically generated by the system.

```
DBE_TASK.ID_SUBMIT(
id IN BIGINT,
what IN TEXT,
next_time IN TIMESTAMP DEFAULT sysdate,
interval_time IN TEXT DEFAULT 'null');
```

Example:

```
gaussdb=# CALL dbe_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
id_submit
```

```

(1 row)
```

- **DBE\_TASK.CANCEL**

The stored procedure CANCEL deletes a specified task.

The prototype of the DBE\_TASK.CANCEL function is as follows:

```
CANCEL(id IN INTEGER);
```

**Table 10-217** DBE\_TASK.CANCEL parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description           |
|-----------|---------|------------------------|--------------|-----------------------|
| id        | Integer | IN                     | No           | Specifies the job ID. |

Example:

```
gaussdb=# CALL dbe_task.cancel(101);
cancel
```

```

(1 row)
```

- **DBE\_TASK.RUN**

The stored procedure runs a scheduled task.

The prototype of the DBE\_TASK.RUN function is as follows:

```
DBE_TASK.RUN(
job IN BIGINT,
force IN BOOLEAN DEFAULT FALSE);
```

**Table 10-218** DBE\_TASK.RUN parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                         |
|-----------|---------|------------------------|--------------|-------------------------------------|
| job       | bigint  | IN                     | No           | Specifies the job ID.               |
| force     | Boolean | IN                     | Yes          | Used only for syntax compatibility. |

Example:

```
gaussdb=# BEGIN
gaussdb$$ DBE_TASK.ID_SUBMIT(12345, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
gaussdb$$ DBE_TASK.RUN(12345);
gaussdb$$ END;
gaussdb$$ /
ANONYMOUS BLOCK EXECUTE
```

- DBE\_TASK.FINISH

The stored procedure FINISH disables or enables a scheduled task.

The prototype of the DBE\_TASK.FINISH function is as follows:

```
DBE_TASK.FINISH(
id IN INTEGER,
broken IN BOOLEAN,
next_time IN TIMESTAMP DEFAULT sysdate);
```

**Table 10-219** DBE\_TASK.FINISH parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                |
|-----------|---------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id        | Integer | IN                     | No           | Specifies the job ID.                                                                                                                                                                                                                      |
| broken    | Boolean | IN                     | No           | Specifies the status flag, <b>true</b> for broken and <b>false</b> for not broken. The current job is updated based on the parameter value <b>true</b> or <b>false</b> . If the parameter is left empty, the job status remains unchanged. |

| Parameter | Type      | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|-----------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| next_time | Timestamp | IN                     | Yes          | Specifies the next execution time. The default value is the current system time. If <b>broken</b> is set to <b>true</b> , <b>next_time</b> is updated to '4000-1-1'. If <b>broken</b> is set to <b>false</b> and <b>next_time</b> is not empty, <b>next_time</b> is updated for the job. If <b>next_time</b> is empty, it will not be updated. This parameter can be omitted, and its default value will be used in this case. |

Example:

```
gaussdb=# CALL dbe_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
id_submit
```

-----  
(1 row)

```
gaussdb=# CALL dbe_task.finish(101, true);
finish
```

-----  
(1 row)

```
gaussdb=# CALL dbe_task.finish(101, false, sysdate);
finish
```

-----  
(1 row)

- **DBE\_TASK.UPDATE**

The stored procedure UPDATE modifies user-definable attributes of a task, including the task content, next-execution time, and execution interval.

The prototype of the DBE\_TASK.UPDATE function is as follows:

```
dbe_task.UPDATE(
id IN INTEGER,
content IN TEXT,
next_time IN TIMESTAMP,
interval_time IN TEXT);
```

**Table 10-220** DBE\_TASK.UPDATE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description           |
|-----------|---------|------------------------|--------------|-----------------------|
| id        | Integer | IN                     | No           | Specifies the job ID. |

| Parameter     | Type      | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|-----------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| content       | Text      | IN                     | Yes          | Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left empty, the system does not update the <b>content</b> parameter for the specified job. Otherwise, the system updates the <b>content</b> parameter for the specified job.                                                                                                                                                                                       |
| next_time     | Timestamp | IN                     | Yes          | Specifies the next execution time. If this parameter is left empty, the system does not update the <b>next_time</b> parameter for the specified job. Otherwise, the system updates the <b>next_time</b> parameter for the specified job.                                                                                                                                                                                                                                    |
| interval_time | Text      | IN                     | Yes          | Specifies the time expression for calculating the next time the job will be executed. If this parameter is left empty, the <b>interval_time</b> parameter is not updated for the specified job. If not, whether the <b>interval_time</b> parameter is valid time or interval type is checked. If it is, it is updated for the specified job. If this parameter is set to <b>null</b> , the job will be executed only once, and the job status will change to 'd' afterward. |

Example:

```
gaussdb=# CALL dbe_task.update(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');
update

(1 row)

gaussdb=# CALL dbe_task.update(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate + 1.0/1440');
update

(1 row)
```

- **DBE\_TASK.CHANGE**

The stored procedure UPDATE modifies user-definable attributes of a task, including the task content, next-execution time, and execution interval.

The prototype of the DBE\_TASK.CHANGE function is as follows:

```
DBE_TASK.CHANGE(
job IN INTEGER,
```

```

what IN TEXT DEFAULT NULL,
next_date IN TIMESTAMP DEFAULT NULL,
job_interval IN TEXT DEFAULT NULL,
instance IN INTEGER DEFAULT NULL,
force IN BOOLEAN DEFAULT false);

```

**Table 10-221** DBE\_TASK.CHANGE parameters

| Parameter    | Type      | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------|-----------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| job          | Integer   | IN                     | No           | Specifies the job ID.                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| what         | Text      | IN                     | Yes          | Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left empty, the system does not update the <b>what</b> parameter for the specified job. Otherwise, the system updates the <b>what</b> parameter for the specified job.                                                                                                                                                                                 |
| next_date    | Timestamp | IN                     | Yes          | Specifies the next execution time. If this parameter is left empty, the system does not update the <b>next_date</b> parameter for the specified job. Otherwise, the system updates the <b>next_date</b> parameter for the specified job.                                                                                                                                                                                                                        |
| job_interval | Text      | IN                     | Yes          | Specifies the time expression for calculating the next time the job will be executed. If this parameter is left empty, the system does not update the <b>job_interval</b> parameter for the specified job. Otherwise, the system updates the <b>job_interval</b> parameter for the specified job after necessary validity check. If this parameter is set to <b>null</b> , the job will be executed only once, and the job status will change to 'd' afterward. |
| instance     | Integer   | IN                     | Yes          | Used only for syntax compatibility.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| force        | Boolean   | IN                     | No           | Used only for syntax compatibility.                                                                                                                                                                                                                                                                                                                                                                                                                             |

Example:

```

gaussdb=# BEGIN
gaussdb$# DBE_TASK.CHANGE(

```

```
gaussdb$# job => 101,
gaussdb$# what => 'insert into t2 values (2);'
gaussdb$#);
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- **DBE\_TASK.CONTENT**

The stored procedure **CONTENT** modifies the procedures to be executed by a specified task.

The prototype of the **DBE\_TASK.CONTENT** function is as follows:

```
DBE_TASK.CONTENT(
id IN INTEGER,
content IN TEXT);
```

**Table 10-222** DBE\_TASK.CONTENT parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                         |
|-----------|---------|------------------------|--------------|-----------------------------------------------------------------------------------------------------|
| id        | Integer | IN                     | No           | Specifies the job ID.                                                                               |
| content   | Text    | IN                     | No           | Specifies the name of the stored procedure, SQL statement block, or program block that is executed. |

 **NOTE**

- If the value specified by the **content** parameter is one or multiple executable SQL statements, program blocks, or stored procedures, this procedure can be executed successfully; otherwise, it will fail to be executed.
- If the value specified by the **content** parameter is a simple statement such as **INSERT** and **UPDATE**, a schema name must be added in front of the table name.

Example:

```
gaussdb=# CALL dbe_task.content(101, 'call userproc();');
content

(1 row)

gaussdb=# CALL dbe_task.content(101, 'insert into tbl_a values(sysdate);');
content

(1 row)
```

- **DBE\_TASK.NEXT\_TIME**

The stored procedure **NEXT\_TIME** modifies the next-execution time attribute of a task.

The prototype of the **DBE\_TASK.NEXT\_TIME** function is as follows:

```
DBE_TASK.NEXT_TIME(
id IN BIGINT,
next_time IN TEXT);
```

**Table 10-223** DBE\_TASK.NEXT\_TIME parameters

| Parameter | Type   | Input/Output Parameter | Can Be Empty | Description                        |
|-----------|--------|------------------------|--------------|------------------------------------|
| id        | bigint | IN                     | No           | Specifies the job ID.              |
| next_time | Text   | IN                     | No           | Specifies the next execution time. |

 **NOTE**

If the specified **next\_time** value is earlier than the current date, the job is executed once immediately.

Example:

```
gaussdb=# CALL dbe_task.next_time(101, sysdate);
next_time

(1 row)
```

- **DBE\_TASK.INTERVAL**

The stored procedure INTERVAL modifies the execution interval attribute of a task.

The prototype of the DBE\_TASK.INTERVAL function is as follows:

```
DBE_TASK.INTERVAL(
id IN INTEGER,
interval_time IN TEXT);
```

**Table 10-224** DBE\_TASK.INTERVAL parameters

| Parameter     | Type    | Input / Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                        |
|---------------|---------|--------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id            | Integer | IN                       | No           | Specifies the job ID.                                                                                                                                                                                                                                                                              |
| interval_time | Text    | IN                       | Yes          | Specifies the time expression for calculating the next time the job will be executed. If this parameter is left empty or set to <b>null</b> , the job will be executed only once, and the job status will change to 'd' afterward. The job interval must be of a valid time type or interval type. |

Example:

```
gaussdb=# CALL dbe_task.interval(101, 'sysdate + 1.0/1440');
interval

(1 row)

gaussdb=# CALL dbe_task.cancel(101);
cancel

(1 row)
```

#### NOTE

For a job that is currently running (that is, **job\_status** is 'r'), it is not allowed to use **cancel**, **update**, **next\_time**, **content**, or **interval** to delete or modify job parameters.

## Constraints

1. After a job is created by using **SUMMIT/ID\_SUBMIT**, the job belongs to the current coordinator (that is, the job is scheduled and executed only on the current coordinator). Other coordinators do not schedule and execute the job. If the coordinator node is faulty, the job cannot be properly executed. You are advised to use the **PKG\_SERVICE.SUBMIT\_ON\_NODES** API to specify the job execution node as CCN to ensure that the job is still available when a node is faulty. Not all coordinators can query, modify, and delete tasks created on other CNs.
2. You can create, update, and delete tasks only using the procedures provided by the **DBE\_TASK** package. These procedures synchronize task information between different CNs and associate primary keys between **pg\_job** and **pg\_job\_proc**. If you use DML statements to add, delete, or modify records in **pg\_job**, task information will become inconsistent between CNs and system catalogs may fail to be associated, compromising internal task management.
3. Each task created by a user is bound to a CN. If the CN fails while a task is being executed, the task status cannot be updated in real time and will stay at 'r'. The task status will be updated to 's' only after the CN recovers. When a CN fails, all tasks on this CN cannot be scheduled or executed until the CN is restored manually, or deleted and then replaced.
4. For each task, the bound CN updates the real-time task information (including the task status, last execution start time, last execution end time, next execution start time, the number of execution failures [if any]) to **pg\_job**, and synchronizes the information to other CNs, ensuring consistent task information between different CNs. In the case of faults on other CNs, task information synchronization is reattempted by the bound CN, which increases job execution time. Although task information fails to be synchronized between CNs, task information can still be properly updated in **pg\_job** on the bound CN and the task can be executed successfully. After the faulty CN recovers, task information such as task execution time and status in its **pg\_job** may be incorrect and will be updated only after the task is executed again on the bound CN.
5. For each job, a thread is established to execute it. If multiple jobs are triggered concurrently as scheduled, the system will need some time to start the required threads, resulting in a latency of 0.1 ms in job execution.

## 10.11.2.6 DBE\_UTILITY

### APIs

Table 10-225 provides all APIs supported by the **DBE\_UTILITY** package.

Table 10-225 DBE\_UTILITY

| API                                                | Description                                                                                                                |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_UTILITY.FORMAT_ERROR_BACKTRACE</a> | Outputs the call stack of an abnormal stored procedure.                                                                    |
| <a href="#">DBE_UTILITY.FORMAT_ERROR_STACK</a>     | Outputs detailed information about a stored procedure exception.                                                           |
| <a href="#">DBE_UTILITY.FORMAT_CALL_STACK</a>      | Outputs the call stack of a stored procedure.                                                                              |
| <a href="#">DBE_UTILITY.GET_TIME</a>               | Outputs the current time, which is used to obtain the execution duration.                                                  |
| <a href="#">DBE_UTILITY.CANONICALIZE</a>           | Canonicalizes the character string of a table name.                                                                        |
| <a href="#">DBE_UTILITY.COMMA_TO_TABLE</a>         | Converts a comma-delimited string of names into a PL/SQL table of names.                                                   |
| <a href="#">DBE_UTILITY.DB_VERSION</a>             | Returns the version number and compatibility version number of the database.                                               |
| <a href="#">DBE_UTILITY.EXEC_DDL_STATEMENT</a>     | Executes DDL statements entered by users.                                                                                  |
| <a href="#">DBE_UTILITY.EXPAND_SQL_TEXT_PROC</a>   | Expands the view of the SQL query.                                                                                         |
| <a href="#">DBE_UTILITY.GET_CPU_TIME</a>           | Returns the measured value of the current CPU processing time.                                                             |
| <a href="#">DBE_UTILITY.GET_ENDIANNESS</a>         | Obtains the big-endian and little-endian information of the byte order on the platform where the database is located.      |
| <a href="#">DBE_UTILITY.GET_HASH_VALUE</a>         | Returns the hash value of a given string.                                                                                  |
| <a href="#">DBE_UTILITY.GET_SQL_HASH</a>           | Outputs the hash value of a given string. This stored procedure is used when <b>proc_outparam_override</b> is not enabled. |
| <a href="#">DBE_UTILITY.IS_BIT_SET</a>             | Checks whether parameter <b>n</b> exists in <b>r</b> .                                                                     |
| <a href="#">DBE_UTILITY.IS_CLUSTER_DATABASE</a>    | Determines whether the current database is running in database cluster mode.                                               |

| API                                                     | Description                                                                                                  |
|---------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_UTILITY.NAME_RE<br/>SOLVE</a>           | Parses the given object name, including synonym translation and necessary authorization checks.              |
| <a href="#">DBE_UTILITY.NAME_T<br/>OKENIZE</a>          | Parses the name in the <b>a [ . b [ . c ] ] [ @ dblink ]</b> format.                                         |
| <a href="#">DBE_UTILITY.OLD_CUR<br/>RENT_SCHEMA</a>     | Returns the name of the database schema in the current user environment.                                     |
| <a href="#">DBE_UTILITY.OLD_CUR<br/>RENT_USER</a>       | Returns the name of the current user.                                                                        |
| <a href="#">DBE_UTILITY.TABLE_TO<br/>_COMMA</a>         | Converts a PL/SQL table of names into a comma-delimited string of names.                                     |
| <a href="#">DBE_UTILITY.GET_SQL_<br/>HASH_FUNC</a>      | Equivalent to DBE_UTILITY.GET_SQL_HASH. This function is used when <b>proc_outparam_override</b> is enabled. |
| DBE_UTILITY.EXPAND_S<br>QL_TEXT                         | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.CANONIC<br>ALIZE_RET                        | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.COMMA_T<br>O_TABLE_FUN                      | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.COMPILE_<br>SCHEMA                          | This is an internal function and is deprecated. You are advised not to use this function.                    |
| DBE_UTILITY.NAME_SEP<br>ARATE                           | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.NAME_TO<br>KENIZE_FUNC                      | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.NAME_TO<br>KENIZE_LOWER                     | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.NAME_TO<br>KENIZE_LOWER_FUNC                | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.PRIVILEGE<br>_CHECK                         | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.SEARCH_C<br>LASS_WITH_NSPOID_O<br>NAME_TYPE | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.SEARCH_O<br>BJECTS                          | This is an internal function and is not recommended.                                                         |
| DBE_UTILITY.SEARCH_O<br>BJECTS_SYNONYM_FILL<br>_SCHEMA  | This is an internal function and is not recommended.                                                         |

| API                                                   | Description                                          |
|-------------------------------------------------------|------------------------------------------------------|
| DBE_UTILITY.SEARCH_PROCEDURE_WITH_NAMESPACE_OID_ONAME | This is an internal function and is not recommended. |
| DBE_UTILITY.SEARCH_SYNONYM_WITH_NAMESPACE_OID_ONAME   | This is an internal function and is not recommended. |
| DBE_UTILITY.TABLE_TO_COMMA_FUNC                       | This is an internal function and is not recommended. |
| DBE_UTILITY.USER_NAME                                 | This is an internal function and is not recommended. |

- DBE\_UTILITY.FORMAT\_ERROR\_BACKTRACE**

Returns the call stack where an error occurs during execution. The prototype of the DBE\_UTILITY.FORMAT\_ERROR\_BACKTRACE function is as follows:

```
DBE_UTILITY.FORMAT_ERROR_BACKTRACE()
RETURN TEXT;
```
- DBE\_UTILITY.FORMAT\_ERROR\_STACK**

Returns the detailed information about the error location when an error occurs during the execution. The prototype of the DBE\_UTILITY.FORMAT\_ERROR\_STACK function is as follows:

```
DBE_UTILITY.FORMAT_ERROR_STACK()
RETURN TEXT;
```
- DBE\_UTILITY.FORMAT\_CALL\_STACK**

Sets the call stack of the output function. The prototype of the DBE\_UTILITY.FORMAT\_CALL\_STACK function is as follows:

```
DBE_UTILITY.FORMAT_CALL_STACK()
RETURN TEXT;
```
- DBE\_UTILITY.GET\_TIME**

Sets the output time, which is usually used for difference. A separate return value is meaningless. The prototype of the DBE\_UTILITY.GET\_TIME function is as follows:

```
DBE_UTILITY.GET_TIME()
RETURN BIGINT;
```
- DBE\_UTILITY.CANONICALIZE**

Canonicalizes the character string of a table name. The procedure handles a single reserved word or keyword, and removes white spaces for a single identifier so that "table" becomes TABLE. The prototype of the DBE\_UTILITY.CANONICALIZE function is as follows:

```
DBE_UTILITY.CANONICALIZE(
 name IN VARCHAR2,
 canon_name OUT VARCHAR2,
 canon_len IN BINARY_INTEGER DEFAULT 1024
);
```

**Table 10-226** DBE\_UTILITY.CANONICALIZE parameters

| Parameter  | Type           | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                         |
|------------|----------------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name       | VARCHAR2       | IN                     | No           | Character string to be canonicalized.                                                                                                                                                                               |
| canon_name | VARCHAR2       | OUT                    | Yes          | Canonicalized character string.                                                                                                                                                                                     |
| canon_len  | BINARY_INTEGER | IN                     | Yes          | Length of the string to be canonicalized, in bytes. If the value of this parameter is less than the actual length (in bytes) of the character string to be standardized, the character string is truncated by byte. |

- DBE\_UTILITY.COMMA\_TO\_TABLE

Converts a comma-delimited string of names into a PL/SQL table of names. The prototype of the DBE\_UTILITY.COMMA\_TO\_TABLE function is as follows:

```
DBE_UTILITY.COMMA_TO_TABLE (
 list IN VARCHAR2,
 tablen OUT BINARY_INTEGER,
 tab OUT VARCHAR2[]
);
```

**Table 10-227** DBE\_UTILITY.COMMA\_TO\_TABLE parameters

| Parameter | Type           | Input/Output Parameter | Can Be Empty | Description                               |
|-----------|----------------|------------------------|--------------|-------------------------------------------|
| list      | VARCHAR2       | IN                     | No           | A comma-delimited string of names.        |
| tablen    | BINARY_INTEGER | OUT                    | Yes          | Number of names in the table.             |
| tab       | VARCHAR2       | OUT                    | Yes          | Table which contains the string of names. |

- DBE\_UTILITY.DB\_VERSION

Returns the version number and compatibility version number of the database. The prototype of the DBE\_UTILITY.DB\_VERSION function is as follows:

```
DBE_UTILITY.DB_VERSION (
 version OUT VARCHAR2
);
```

**Table 10-228** DBE\_UTILITY.DB\_VERSION parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                                                                                                |
|-----------|----------|------------------------|--------------|------------------------------------------------------------------------------------------------------------|
| version   | VARCHAR2 | OUT                    | No           | Output parameter, which indicates the internal database software version. The value is a character string. |

- DBE\_UTILITY.EXEC\_DDL\_STATEMENT

Executes DDL statements entered by users. The prototype of the DBE\_UTILITY.EXEC\_DDL\_STATEMENT function is as follows:

```
DBE_UTILITY.EXEC_DDL_STATEMENT (
 parse_string IN TEXT
);
```

**Table 10-229** DBE\_UTILITY.EXEC\_DDL\_STATEMENT parameters

| Parameter    | Type | Input/Output Parameter | Can Be Empty | Description                    |
|--------------|------|------------------------|--------------|--------------------------------|
| parse_string | TEXT | IN                     | Yes          | DDL statements to be executed. |

- DBE\_UTILITY.EXPAND\_SQL\_TEXT\_PROC

Expands the view of the SQL query. It recursively expands the view objects in the view until a table is displayed. The function prototype of DBE\_UTILITY.EXPAND\_SQL\_TEXT\_PROC is as follows:

```
DBE_UTILITY.EXPAND_SQL_TEXT_PROC (
 input_sql_text IN CLOB,
 output_sql_text OUT CLOB
);
```

**Table 10-230** DBE\_UTILITY.EXPAND\_SQL\_TEXT\_PROC parameters

| Parameter       | Type         | Input/Output Parameter | Can Be Empty | Description                           |
|-----------------|--------------|------------------------|--------------|---------------------------------------|
| input_sql_text  | CL<br>O<br>B | IN                     | No           | Input SQL text.                       |
| output_sql_text | CL<br>O<br>B | O<br>U<br>T            | No           | Output SQL text of the expanded view. |

 **NOTE**

In the **input\_sql\_text** parameter entered by a user, a schema prefix must be added to the object in the SQL statement. Otherwise, the function reports an error indicating that no object is found. If **set behavior\_compat\_options** is set to **bind\_procedure\_searchpath**, you do not need to forcibly specify the schema prefix.

- **DBE\_UTILITY.GET\_CPU\_TIME**

Returns the measured value of the current CPU processing time, in hundredths of a second. The prototype of the DBE\_UTILITY.GET\_CPU\_TIME function is as follows:

```
DBE_UTILITY.GET_CPU_TIME()
RETURN NUMBER;
```

- **DBE\_UTILITY.GET\_ENDIANNES**

Obtains the big-endian and little-endian information of the byte order on the platform where the database is located. DBE\_UTILITY. The prototype of the GET\_ENDIANNES function is as follows:

```
DBE_UTILITY.GET_ENDIANNES
RETURN INTEGER;
```

- **DBE\_UTILITY.GET\_HASH\_VALUE**

Returns the hash value of a given string. The prototype of the DBE\_UTILITY.GET\_HASH\_VALUE function is as follows:

```
DBE_UTILITY.GET_HASH_VALUE(
 name IN VARCHAR2(n),
 base IN INTEGER,
 hash_size IN INTEGER)
RETURN INTEGER;
```

**Table 10-231** DBE\_UTILITY.GET\_HASH\_VALUE parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                                         |
|-----------|----------|------------------------|--------------|-----------------------------------------------------|
| name      | VARCHAR2 | IN                     | No           | Character string to be hashed.                      |
| base      | INTEGER  | IN                     | No           | Start value of the returned hash value.             |
| hash_size | INTEGER  | IN                     | No           | Size of the hash table to which the hash is mapped. |

- **DBE\_UTILITY.GET\_SQL\_HASH**

Outputs the hash value of a given character string using the MD5 algorithm. The prototype of the DBE\_UTILITY.GET\_SQL\_HASH function is as follows:

```
DBE_UTILITY.GET_SQL_HASH(
 name IN VARCHAR2,
 hash OUT RAW,
 last4bytes OUT BIGINT
);
```

**Table 10-232** DBE\_UTILITY.GET\_SQL\_HASH parameters

| Parameter  | Type     | Input/Output Parameter | Can Be Empty | Description                                                                       |
|------------|----------|------------------------|--------------|-----------------------------------------------------------------------------------|
| name       | VARCHAR2 | IN                     | No           | Character string to be hashed.                                                    |
| hash       | RAW      | OUTPUT                 | No           | Complete hexadecimal MD5 hash value.                                              |
| last4bytes | BIGINT   | OUTPUT                 | No           | Last four bytes of the MD5 hash value, which is displayed as an unsigned integer. |

 **NOTE**

After **set behavior\_compat\_options** is set to a parameter other than **proc\_outparam\_override** (for parameter settings, contact an administrator), call the DBE\_UTILITY.GET\_SQL\_HASH function. If you call the DBE\_UTILITY.GET\_SQL\_HASH\_FUNC function, the value fails to be assigned.

- DBE\_UTILITY.IS\_BIT\_SET

Checks whether parameter **n** exists in **r**. DBE\_UTILITY. The prototype of the IS\_BIT\_SET function is as follows:

```
DBE_UTILITY.IS_BIT_SET (
 r IN RAW,
 n IN INTEGER)
RETURN INTEGER;
```

**Table 10-233** DBE\_UTILITY.IS\_BIT\_SET parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                               |
|-----------|---------|------------------------|--------------|-----------------------------------------------------------|
| r         | RAW     | IN                     | No           | 4 bytes plus the actual hexadecimal string.               |
| n         | INTEGER | IN                     | No           | Determines whether the value exists in the binary system. |

- **DBE\_UTILITY.IS\_CLUSTER\_DATABASE**

Determines whether the current database is running in database cluster mode. The prototype of the DBE\_UTILITY.IS\_CLUSTER\_DATABASE function is as follows:

```
DBE_UTILITY.IS_CLUSTER_DATABASE
RETURN BOOLEAN;
```

- **DBE\_UTILITY.NAME\_RESOLVE**

Parses the given object name, including synonym translation and necessary authorization checks. The prototype of the DBE\_UTILITY.NAME\_RESOLVE function is as follows:

```
DBE_UTILITY.NAME_RESOLVE (
name IN VARCHAR2,
context IN INTEGER,
schema OUT VARCHAR2,
part1 OUT VARCHAR2,
part2 OUT VARCHAR2,
dblink OUT VARCHAR2,
part1_type OUT INTEGER,
object_number OUT OID
);
```

**Table 10-234** DBE\_UTILITY.NAME\_RESOLVE parameters

| Parameter  | Type     | Input/Output Parameter | Can Be Empty | Description                                                                                                                                         |
|------------|----------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| name       | VARCHAR2 | IN                     | No           | Name of the object to be parsed. The structure is [[a.]b.]c[@d].                                                                                    |
| context    | INTEGER  | IN                     | No           | Start value of the returned hash value.                                                                                                             |
| schema     | VARCHAR2 | OUT                    | No           | Schema of an object.                                                                                                                                |
| part1      | VARCHAR2 | OUT                    | No           | First part of the name. The type of this column is specified by <b>part1_type</b> .                                                                 |
| part2      | VARCHAR2 | OUT                    | Yes          | If this column is not empty, the value is the subprogram name.                                                                                      |
| dblink     | VARCHAR2 | OUT                    | Yes          | Database link.                                                                                                                                      |
| part1_type | INTEGER  | OUT                    | No           | Part 1 types: <ul style="list-style-type: none"> <li>● 5: synonym</li> <li>● 7: procedure (top level)</li> <li>● 8: function (top level)</li> </ul> |

| Parameter     | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                         |
|---------------|------|------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object_number | OID  | OUT                    | No           | Object ID. In ORA-compatible database, <b>object_number</b> is of the numeric type, indicating the object ID. In GaussDB, <b>object_number</b> is of the OID type and does not support implicit conversion from a number to an OID. |

- DBE\_UTILITY.NAME\_TOKENIZE

Parses names in the **a [. b [. c ]][@ dblink ]** format. If a name contains double quotation marks, the double quotation marks are deleted. Otherwise, the name becomes uppercase letters. The prototype of the DBE\_UTILITY.NAME\_TOKENIZE function is as follows:

```
DBE_UTILITY.NAME_TOKENIZE (
name IN VARCHAR2,
a OUT VARCHAR2,
b OUT VARCHAR2,
c OUT VARCHAR2,
dblink OUT VARCHAR2,
nextpos OUT INTEGER
);
```

**Table 10-235** DBE\_UTILITY.NAME\_TOKENIZE parameters

| Parameter | Type     | Input/Output Parameter | Can Be Empty | Description                                                                  |
|-----------|----------|------------------------|--------------|------------------------------------------------------------------------------|
| name      | VARCHAR2 | IN                     | No           | Name, consisting of SQL identifiers (for example, <b>scott.foo@dblink</b> ). |
| a         | VARCHAR2 | OUTPUT                 | No           | First token of the name.                                                     |
| b         | VARCHAR2 | OUTPUT                 | Yes          | Second token of the name.                                                    |
| c         | VARCHAR2 | OUTPUT                 | Yes          | Third token of the name.                                                     |
| dblink    | VARCHAR2 | OUTPUT                 | Yes          | Database link.                                                               |
| nextpos   | INTEGER  | OUTPUT                 | No           | Next position of a parsed character string.                                  |

- **DBE\_UTILITY.OLD\_CURRENT\_SCHEMA**  
Returns the name of the database schema in the current user environment. The prototype of the DBE\_UTILITY.OLD\_CURRENT\_SCHEMA function is as follows:

```
DBE_UTILITY.OLD_CURRENT_SCHEMA()
RETURN VARCHAR;
```

- DBE\_UTILITY.OLD\_CURRENT\_USER

Returns the name of the current user. The prototype of the DBE\_UTILITY.OLD\_CURRENT\_USER function is as follows:

```
DBE_UTILITY.OLD_CURRENT_USER()
RETURN TEXT;
```

- DBE\_UTILITY.TABLE\_TO\_COMMA

Converts a PL/SQL table of names into a comma-delimited string of names. The prototype of the DBE\_UTILITY.TABLE\_TO\_COMMA function is as follows:

```
DBE_UTILITY.TABLE_TO_COMMA (
 tab IN VARCHAR2[],
 tablen OUT BINARY_INTEGER,
 list OUT VARCHAR2
);
```

**Table 10-236** DBE\_UTILITY.TABLE\_TO\_COMMA parameters

| Parameter | Type           | Input/Output Parameter | Can Be Empty | Description                                      |
|-----------|----------------|------------------------|--------------|--------------------------------------------------|
| tab       | VARCHAR2[]     | IN                     | No           | Structure table array that contains table names. |
| tablen    | BINARY_INTEGER | OUT                    | No           | Number of tables in a structure table.           |
| list      | VARCHAR2       | OUT                    | No           | A comma-delimited string of names.               |

- DBE\_UTILITY.GET\_SQL\_HASH\_FUNC

Uses the MD5 algorithm to output the hash value of a given character string.  
The function prototype of DBE\_UTILITY.GET\_SQL\_HASH\_FUNC is:

```
DBE_UTILITY.GET_SQL_HASH_FUNC(
 name IN VARCHAR2,
 hash OUT RAW,
 last4bytes OUT BIGINT
);
```

**Table 10-237** DBE\_UTILITY.GET\_SQL\_HASH\_FUNC parameters

| Parameter  | Type     | Input/Output Parameter | Can Be Empty | Description                                                                       |
|------------|----------|------------------------|--------------|-----------------------------------------------------------------------------------|
| name       | VARCHAR2 | IN                     | No           | Character string to be hashed.                                                    |
| hash       | RAW      | OUT                    | No           | Complete hexadecimal MD5 hash value.                                              |
| last4bytes | BIGINT   | OUT                    | No           | Last four bytes of the MD5 hash value, which is displayed as an unsigned integer. |

 **NOTE**

After setting **behavior\_compat\_options** to 'proc\_outparam\_override', call the DBE\_UTILITY.GET\_SQL\_HASH\_FUNC function. If you call the DBE\_UTILITY.GET\_SQL\_HASH function, a parameter mismatch error is reported.

## Examples

```
CREATE OR REPLACE PROCEDURE test_get_time1()
AS
declare
 start_time bigint;
 end_time bigint;
BEGIN
 start_time:= dbe_utility.get_time ();
 pg_sleep(1);
 end_time:=dbe_utility.get_time ();
 dbe_output.print_line(end_time - start_time);
END;
/
CREATE PROCEDURE
```

```
-- Canonicalize the character string of a table name.
declare
 cname varchar2(50);
begin
 dbe_utility.canonicalize('seg1', cname, 50);
 dbe_output.put_line(cname);
end;
/
-- The expected result is as follows:
SEG1
ANONYMOUS BLOCK EXECUTE

-- Convert the input character string into an array of table names.
DECLARE
tab_list VARCHAR2(100) := 't1,t2';
len BINARY_INTEGER;
tab varchar2[];
BEGIN
 dbe_output.put_line('table list is:' || tab_list);
 dbe_utility.comma_to_table(tab_list, len, tab);
END;
/
-- The expected result is as follows:
table list is: t1,t2
ANONYMOUS BLOCK EXECUTE

-- Check the version number and compatibility version number of the database.
declare
 v_version varchar2;
begin
 dbe_utility.db_version(v_version);
 v_version:=left(v_version, 8);
 dbe_output.print_line('version:' || v_version);
end;
/
-- The expected result is as follows:
version:gaussdb
ANONYMOUS BLOCK EXECUTE

-- Check the measured value of the current CPU processing time.
DECLARE
 cputime NUMBER;
BEGIN
 cputime := dbe_utility.get_cpu_time;
 dbe_output.put_line('cpu time:' || cputime);
END;
/
-- The expected result is as follows (the value is not fixed):
cpu time: 70179
ANONYMOUS BLOCK EXECUTE

-- Obtain the big-endian and little-endian information of the byte order on the platform where the
database is located.
BEGIN
 dbe_output.PUT_LINE(dbe_utility.GET_ENDIANNES);
END;
/
-- The expected result is as follows:
2
ANONYMOUS BLOCK EXECUTE

-- Obtain the hash value of a given string.
DECLARE
 result NUMBER(28);
BEGIN
 result := dbe_utility.get_hash_value('hello',10,10);
 dbe_output.put_line(result);
END;
```

```

/
-- The expected result is as follows:
11
ANONYMOUS BLOCK EXECUTE

-- Check whether the current database is in cluster mode.
DECLARE
 is_cluster BOOLEAN;
BEGIN
 is_cluster := dbe_utility.IS_CLUSTER_DATABASE;
 dbe_output.put_line('CLUSTER DATABASE: ' || CASE WHEN is_cluster THEN 'TRUE' ELSE 'FALSE' END);
END;
/
-- The expected result is as follows:
CLUSTER DATABASE: TRUE
ANONYMOUS BLOCK EXECUTE

-- Obtain the name of the database schema in the current user environment.
DECLARE
 schm varchar2(100);
BEGIN
 schm := dbe_utility.old_current_schema;
 dbe_output.put_line('current schema: ' || schm);
END;
/
-- The expected result is as follows (the result is the schema name of the current database, which is not
fixed):
current schema: public
ANONYMOUS BLOCK EXECUTE

-- Obtain the current username.
select dbe_utility.old_current_user from sys_dummy;
-- The expected result is as follows (the result is the username of the current database, which is not fixed):
old_current_user

test
(1 row)

```

### 10.11.2.7 DBE\_SQL

#### APIs

**Table 10-238** lists APIs supported by the **DBE\_SQL** package.

**Table 10-238** DBE\_SQL

| API                                            | Description                                                     |
|------------------------------------------------|-----------------------------------------------------------------|
| <a href="#">DBE_SQL.REGISTER_CONTEXT</a>       | Opens a cursor.                                                 |
| <a href="#">DBE_SQL.SQL_UNREGISTER_CONTEXT</a> | Closes an open cursor.                                          |
| <a href="#">DBE_SQL.SQL_SET_SQL</a>            | Passes a set of SQL statements or anonymous blocks to a cursor. |
| <a href="#">DBE_SQL.SQL_RUN</a>                | Executes SQL statements or anonymous blocks in a given cursor.  |
| <a href="#">DBE_SQL.NEXT_ROW</a>               | Reads a row of cursor data.                                     |
| <a href="#">DBE_SQL.SET_RESULT_TYPE</a>        | Dynamically defines a column.                                   |

| API                                                  | Description                                                                                       |
|------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| <a href="#">DBE_SQL.SET_RESULT_TYPE_CHAR</a>         | Dynamically defines a column of the CHAR type.                                                    |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_INT</a>          | Dynamically defines a column of the INT type.                                                     |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_LONG</a>         | Dynamically defines a column of the LONG type.                                                    |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_RAW</a>          | Dynamically defines a column of the RAW type.                                                     |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_TEXT</a>         | Dynamically defines a column of the TEXT type.                                                    |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_UNKN<br/>OWN</a> | Dynamically defines a column of an unknown type.                                                  |
| <a href="#">DBE_SQL.GET_RESULT</a>                   | Reads a dynamically defined column value.                                                         |
| <a href="#">DBE_SQL.GET_RESULT_CHAR</a>              | Reads a dynamically defined column value of the CHAR type.                                        |
| <a href="#">DBE_SQL.GET_RESULT_INT</a>               | Reads a dynamically defined column value of the INT type.                                         |
| <a href="#">DBE_SQL.GET_RESULT_LONG</a>              | Reads a dynamically defined column value of the LONG type.                                        |
| <a href="#">DBE_SQL.GET_RESULT_RAW</a>               | Reads a dynamically defined column value of the RAW type.                                         |
| <a href="#">DBE_SQL.GET_RESULT_TEXT</a>              | Reads a dynamically defined column value of the TEXT type.                                        |
| <a href="#">DBE_SQL.GET_RESULT_UNKNOWN</a>           | Reads a dynamically defined column value of an unknown type.                                      |
| <a href="#">DBE_SQL.DBE_SQL_GET_RESULT_CH<br/>AR</a> | Reads a dynamically defined column value of the CHAR type.                                        |
| <a href="#">DBE_SQL.DBE_SQL_GET_RESULT_LO<br/>NG</a> | Reads a dynamically defined column value of the LONG type.                                        |
| <a href="#">DBE_SQL.DBE_SQL_GET_RESULT_RA<br/>W</a>  | Reads a dynamically defined column value of the RAW type.                                         |
| <a href="#">DBE_SQL.IS_ACTIVE</a>                    | Checks whether a cursor is opened.                                                                |
| <a href="#">DBE_SQL.LAST_ROW_COUNT</a>               | Returns the cumulative count of obtained rows.                                                    |
| <a href="#">DBE_SQL.RUN_AND_NEXT</a>                 | Reads data of a cursor after a set of dynamically defined operations are performed on the cursor. |

| API                                            | Description                                                       |
|------------------------------------------------|-------------------------------------------------------------------|
| <a href="#">DBE_SQL.SQL_BIND_VARIABLE</a>      | Binds a value to a variable in a statement.                       |
| <a href="#">DBE_SQL.SQL_BIND_ARRAY</a>         | Binds a group of values to a variable in a statement.             |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_INTS</a>   | Dynamically defines a column of the INT array type.               |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_TEXTS</a>  | Dynamically defines a column of the TEXT array type.              |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_RAWS</a>   | Dynamically defines a column of the RAW array type.               |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_BYTES</a>  | Dynamically defines a column of the BYTEA array type.             |
| <a href="#">DBE_SQL.SET_RESULT_TYPE_CHARS</a>  | Dynamically defines a column of the CHAR array type.              |
| <a href="#">DBE_SQL.SET_RESULTS_TYPE</a>       | Dynamically defines a column of the array type.                   |
| <a href="#">DBE_SQL.GET_RESULTS_INT</a>        | Reads a dynamically defined column value of the INT array type.   |
| <a href="#">DBE_SQL.GET_RESULTS_TEXT</a>       | Reads a dynamically defined column value of the TEXT array type.  |
| <a href="#">DBE_SQL.GET_RESULTS_RAW</a>        | Reads a dynamically defined column value of the RAW array type.   |
| <a href="#">DBE_SQL.GET_RESULTS_BYTEA</a>      | Reads a dynamically defined column value of the BYTEA array type. |
| <a href="#">DBE_SQL.GET_RESULTS_CHAR</a>       | Reads a dynamically defined column value of the CHAR array type.  |
| <a href="#">DBE_SQL.GET_RESULTS</a>            | Reads a dynamically defined column value.                         |
| <a href="#">DBE_SQL.DESCRIBE_COLUMNS</a>       | Describes the column information read by the cursor.              |
| <a href="#">DBE_SQL.SQL_DESCRIBE_COLUMNS</a>   | Describes the column information read by the cursor.              |
| <a href="#">DBE_SQL.BIND_VARIABLE</a>          | Binds parameters.                                                 |
| <a href="#">DBE_SQL.SQL_SET_RESULTS_TYPE_C</a> | Dynamically defines a column of the array type.                   |
| <a href="#">DBE_SQL.SQL_GET_VALUES_C</a>       | Reads a dynamically defined column value.                         |

| API                                                   | Description                                                          |
|-------------------------------------------------------|----------------------------------------------------------------------|
| <a href="#">DBE_SQL.GET_VARIABLE_RESULT</a>           | Reads the return value of an SQL statement.                          |
| <a href="#">DBE_SQL.GET_VARIABLE_RESULT_CHARACTER</a> | Reads the return value (of the char type) of an SQL statement.       |
| <a href="#">DBE_SQL.GET_VARIABLE_RESULT_ROW</a>       | Reads the return value (of the row type) of an SQL statement.        |
| <a href="#">DBE_SQL.GET_VARIABLE_RESULT_TEXT</a>      | Reads the return value (of the text type) of an SQL statement.       |
| <a href="#">DBE_SQL.GET_VARIABLE_RESULT_INTEGER</a>   | Reads the return value (of the int type) of an SQL statement.        |
| <a href="#">DBE_SQL.GET_ARRAY_RESULT_TEXT</a>         | Reads the return value (of the text array type) of an SQL statement. |
| <a href="#">DBE_SQL.GET_ARRAY_RESULT_ROW</a>          | Reads the return value (of the row array type) of an SQL statement.  |
| <a href="#">DBE_SQL.GET_ARRAY_RESULT_CHARACTER</a>    | Reads the return value (of the char array type) of an SQL statement. |
| <a href="#">DBE_SQL.GET_ARRAY_RESULT_INTEGER</a>      | Reads the return value (of the int array type) of an SQL statement.  |

 **NOTE**

- You are advised to use **db\_sql.set\_result\_type** and **db\_sql.get\_result** to define columns.
- If the size of the result set is greater than the value of **work\_mem**, the result set will be spilled to a disk temporarily. The value of **work\_mem** must be no greater than 512 MB.
- **DBE\_SQL.REGISTER\_CONTEXT**  
This function opens a cursor, which is the prerequisite for the subsequent **db\_sql** operations. This function does not transfer any parameter. It automatically generates cursor IDs in an ascending order and returns values to integer variables.

 **CAUTION**

Cursors opened by **DBE\_SQL** are session-level variables. Cross-session calling of opened cursors (such as autonomous transactions) is not supported. If a cross-session cursor is called, the behavior is unpredictable.

The prototype of the **DBE\_SQL.REGISTER\_CONTEXT** function is as follows:

```
DBE_SQL.REGISTER_CONTEXT(
)
RETURN INTEGER;
```

- DBE\_SQL.SQL\_UNREGISTER\_CONTEXT

This function closes a cursor, which is the end of each **dbe\_sql** operation. If this function is not called when the stored procedure ends, the memory is still occupied by the cursor. Therefore, remember to close a cursor when you do not need to use it. If an exception occurs, the stored procedure exits but the cursor is not closed. Therefore, you are advised to include this API in the exception handling of the stored procedure.

The prototype of the DBE\_SQL.SQL\_UNREGISTER\_CONTEXT function is as follows:

```
DBE_SQL.SQL_UNREGISTER_CONTEXT(
 context_id IN INTEGER
)
RETURN INTEGER;
```

**Table 10-239** DBE\_SQL.SQL\_UNREGISTER\_CONTEXT parameters

| Parameter  | Description                   |
|------------|-------------------------------|
| context_id | ID of the cursor to be closed |

- DBE\_SQL.SQL\_SET\_SQL

Parses SQL statements or anonymous blocks in a given cursor. The statement parameters can be transferred only through the TEXT type. The length cannot exceed 1 GB.

The prototype of the DBE\_SQL.SQL\_SET\_SQL function is as follows:

```
DBE_SQL.SQL_SET_SQL(
 context_id IN INTEGER,
 query_string IN TEXT,
 language_flag IN INTEGER
)
RETURN BOOLEAN;
```

**Table 10-240** DBE\_SQL.SQL\_SET\_SQL parameters

| Parameter     | Description                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------|
| context_id    | ID of the cursor whose query statement is to be parsed                                                                         |
| query_string  | Query statement to be parsed                                                                                                   |
| language_flag | Version language. The value <b>1</b> indicates an incompatible version, and the value <b>2</b> indicates a compatible version. |

- DBE\_SQL.SQL\_RUN

This function executes a given cursor. This function receives a cursor ID and executes SQL statements or anonymous blocks in a given cursor.

The prototype of the DBE\_SQL.SQL\_RUN function is as follows:

```
DBE_SQL.SQL_RUN(
 context_id IN INTEGER,
)
RETURN INTEGER;
```

**Table 10-241** DBE\_SQL.SQL\_RUN parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor whose query statement is to be parsed |

- DBE\_SQL.NEXT\_ROW

This function returns the number of data rows that meet query conditions. Each time the API is executed, the system obtains a set of new rows until all data is read.

The prototype of the DBE\_SQL.NEXT\_ROW function is as follows:

```
DBE_SQL.NEXT_ROW(
 context_id IN INTEGER,
)
RETURN INTEGER;
```

**Table 10-242** DBE\_SQL.NEXT\_ROW parameters

| Parameter  | Description                     |
|------------|---------------------------------|
| context_id | ID of the cursor to be executed |

- DBE\_SQL.SET\_RESULT\_TYPE

This function defines columns returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE function is as follows:

```
DBE_SQL.SET_RESULT_TYPE(
 context_id IN INTEGER,
 pos IN INTEGER,
 column_ref IN ANYELEMENT,
 maxsize IN INTEGER default 1024
)
RETURN INTEGER;
```

**Table 10-243** DBE\_SQL.SET\_RESULT\_TYPE parameters

| Parameter  | Description                                                                                                    |
|------------|----------------------------------------------------------------------------------------------------------------|
| context_id | ID of the cursor to be executed                                                                                |
| pos        | Relative position of the queried columns in the returned result. The value starts from 1.                      |
| column_ref | Variable of any type. You can select an appropriate API to dynamically define columns based on variable types. |
| maxsize    | Length of the defined column return type.                                                                      |

- DBE\_SQL.SET\_RESULT\_TYPE\_CHAR

This function defines columns of the CHAR type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_CHAR function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_CHAR(
 context_id IN INTEGER,
 pos IN INTEGER,
 column_ref IN TEXT,
 column_size IN INTEGER
)
RETURN INTEGER;
```

**Table 10-244** DBE\_SQL.SET\_RESULT\_TYPE\_CHAR parameters

| Parameter   | Description                                            |
|-------------|--------------------------------------------------------|
| context_id  | ID of the cursor to be executed                        |
| pos         | Position of a dynamically defined column in the query. |
| column_ref  | Parameter to be defined                                |
| column_size | Length of a dynamically defined column                 |

- DBE\_SQL.SET\_RESULT\_TYPE\_INT

This function defines columns of the INT type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_INT function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_INT(
 context_id IN INTEGER,
 pos IN INTEGER
)
RETURN INTEGER;
```

**Table 10-245** DBE\_SQL.SET\_RESULT\_TYPE\_INT parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be executed                        |
| pos        | Position of a dynamically defined column in the query. |

- DBE\_SQL.SET\_RESULT\_TYPE\_LONG

This function defines columns of a long type (not LONG) returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type. The maximum size of a long column is 1 GB.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_LONG function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_LONG(
 context_id IN INTEGER,
 pos IN INTEGER
)
RETURN INTEGER;
```

**Table 10-246** DBE\_SQL.SET\_RESULT\_TYPE\_LONG parameters

| Parameter  | Description                                           |
|------------|-------------------------------------------------------|
| context_id | ID of the cursor to be executed                       |
| pos        | Position of a dynamically defined column in the query |

- **DBE\_SQL.SET\_RESULT\_TYPE\_RAW**

This function defines columns of the RAW type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_RAW function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_RAW(
 context_id IN INTEGER,
 pos IN INTEGER,
 column_ref IN RAW,
 column_size IN INTEGER
)
RETURN INTEGER;
```

**Table 10-247** DBE\_SQL.SET\_RESULT\_TYPE\_RAW parameters

| Parameter   | Description                                            |
|-------------|--------------------------------------------------------|
| context_id  | ID of the cursor to be executed                        |
| pos         | Position of a dynamically defined column in the query. |
| column_ref  | RAW variable                                           |
| column_size | Column length                                          |

- **DBE\_SQL.SET\_RESULT\_TYPE\_TEXT**

This function defines columns of the TEXT type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_TEXT function is as follows:

```
DBE_SQL.DEFINE_COLUMN_CHAR(
 context_id IN INTEGER,
 pos IN INTEGER,
 maxsize IN INTEGER
)
RETURN INTEGER;
```

**Table 10-248** DBE\_SQL.SET\_RESULT\_TYPE\_TEXT parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be executed                        |
| pos        | Position of a dynamically defined column in the query. |
| maxsize    | Maximum length of the defined TEXT type                |

- DBE\_SQL.SET\_RESULT\_TYPE\_UNKNOWN**  
 This function processes columns of unknown data types returned from a given cursor. It is used only for the system to report an error and exist when the type cannot be identified.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_UNKNOWN function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_UNKNOWN(
 context_id IN INTEGER,
 pos IN INTEGER,
 col_type IN TEXT
)
RETURN INTEGER;
```

**Table 10-249** DBE\_SQL.SET\_RESULT\_TYPE\_UNKNOWN parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be executed                        |
| posn       | Position of a dynamically defined column in the query. |
| col_type   | Dynamically defined parameter                          |

- DBE\_SQL.GET\_RESULT**  
 This stored procedure returns the cursor element value in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

The prototype of the DBE\_SQL.GET\_RESULT stored procedure is as follows:

```
DBE_SQL.GET_RESULT(
 context_id IN INTEGER,
 pos IN INTEGER,
 column_value INOUT ANYELEMENT
);
```

**Table 10-250** DBE\_SQL.GET\_RESULT parameters

| Parameter    | Description                                                                               |
|--------------|-------------------------------------------------------------------------------------------|
| context_id   | ID of the cursor to be executed                                                           |
| pos          | Relative position of the queried columns in the returned result. The value starts from 1. |
| column_value | Return value of a defined column                                                          |

- DBE\_SQL.GET\_RESULT\_CHAR

This stored procedure returns the value of the **CHAR** type in a specified position of a cursor and accesses the data obtained by **DBE\_SQL.NEXT\_ROW**.

The prototype of the DBE\_SQL.GET\_RESULT\_CHAR stored procedure is as follows:

```
DBE_SQL.GET_RESULT_CHAR(
 context_id IN INTEGER,
 pos IN INTEGER,
 tr INOUT CHARACTER,
 err INOUT NUMERIC,
 actual_length INOUT INTEGER
);
```

**Table 10-251** DBE\_SQL.GET\_RESULT\_CHAR parameters

| Parameter     | Description                                                                                                                                              |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| context_id    | ID of the cursor to be executed                                                                                                                          |
| pos           | Position of a dynamically defined column in the query.                                                                                                   |
| tr            | Return value                                                                                                                                             |
| err           | Error No. It is an output parameter. The input parameter must be a variable. Currently, the output value is <b>-1</b> regardless of the input parameter. |
| actual_length | Length of a return value                                                                                                                                 |

- DBE\_SQL.GET\_RESULT\_INT

This function returns the value of the **INT** type in a specified position of a cursor and accesses the data obtained by **DBE\_SQL.NEXT\_ROW**. The prototype of the DBE\_SQL.GET\_RESULT\_INT function is as follows:

```
DBE_SQL.GET_RESULT_INT(
 context_id IN INTEGER,
 pos IN INTEGER
)
RETURN INTEGER;
```

**Table 10-252** DBE\_SQL.GET\_RESULT\_INT parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be executed                        |
| pos        | Position of a dynamically defined column in the query. |

- DBE\_SQL.GET\_RESULT\_LONG

This stored procedure returns the value of a long column type (not **LONG** or **BIGINT**) in a specified position of a cursor and accesses the data obtained by **DBE\_SQL.NEXT\_ROW**.

The prototype of the DBE\_SQL.GET\_RESULT\_LONG stored procedure is as follows:

```
DBE_SQL.GET_RESULT_LONG(
 context_id IN INTEGER,
 pos IN INTEGER,
 lgth IN INTEGER,
 off_set IN INTEGER,
 vl INOUT TEXT,
 vl_length INOUT INTEGER
);
```

**Table 10-253** DBE\_SQL.GET\_RESULT\_LONG parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be executed                        |
| pos        | Position of a dynamically defined column in the query. |
| lgth       | Length of a return value                               |
| off_set    | Start position of a return value                       |
| vl         | Return value                                           |
| vl_length  | Length of a return value                               |

- **DBE\_SQL.GET\_RESULT\_RAW**  
Returns the value of the RAW type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

The prototype of the DBE\_SQL.GET\_RESULT\_RAW stored procedure is as follows:

```
DBE_SQL.GET_RESULT_RAW(
 context_id IN INTEGER,
 pos IN INTEGER,
 tr INOUT RAW,
 err INOUT NUMERIC,
 actual_length INOUT INTEGER
);
```

**Table 10-254** DBE\_SQL.GET\_RESULT\_RAW parameters

| Parameter  | Description                                                                                                                                                  |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| context_id | ID of the cursor to be executed.                                                                                                                             |
| pos        | Position of a dynamically defined column in the query.                                                                                                       |
| tr         | Returned column value.                                                                                                                                       |
| err        | Error number. It is an output parameter. The input parameter must be a variable. Currently, the output value is <b>-1</b> regardless of the input parameter. |

| Parameter     | Description                                                                    |
|---------------|--------------------------------------------------------------------------------|
| actual_length | Length of a return value. The value longer than this length will be truncated. |

- DBE\_SQL.GET\_RESULT\_TEXT

This function returns the value of the TEXT type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

The prototype of the DBE\_SQL.GET\_RESULT\_TEXT function is as follows:

```
DBE_SQL.GET_RESULT_TEXT(
 context_id IN INTEGER,
 pos IN INTEGER
)
RETURN TEXT;
```

**Table 10-255** DBE\_SQL.GET\_RESULT\_TEXT parameters

| Parameter  | Description                                           |
|------------|-------------------------------------------------------|
| context_id | ID of the cursor to be executed                       |
| pos        | Position of a dynamically defined column in the query |

- DBE\_SQL.GET\_RESULT\_UNKNOWN

This function returns the value of an unknown type in a specified position of a cursor. It serves as an error handling API when the type is not unknown.

The prototype of the DBE\_SQL.GET\_RESULT\_UNKNOWN function is as follows:

```
DBE_SQL.GET_RESULT_UNKNOWN(
 context_id IN INTEGER,
 pos IN INTEGER,
 col_type IN TEXT
)
RETURN TEXT;
```

**Table 10-256** DBE\_SQL.GET\_RESULT\_UNKNOWN parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be executed                        |
| pos        | Position of a dynamically defined column in the query. |
| col_type   | Returned parameter type                                |

- DBE\_SQL.DBE\_SQL\_GET\_RESULT\_CHAR

This function returns the value of the CHAR type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW. Different from DBE\_SQL.GET\_RESULT\_CHAR, the length of the return value is not set and the entire string is returned.

The prototype of the DBE\_SQL.DBE\_SQL\_GET\_RESULT\_CHAR function is as follows:

```
DBE_SQL.DBE_SQL_GET_RESULT_CHAR(
 context_id IN INTEGER,
 pos IN INTEGER
)
RETURN CHARACTER;
```

**Table 10-257** DBE\_SQL.DBE\_SQL\_GET\_RESULT\_CHAR parameters

| Parameter  | Description                                           |
|------------|-------------------------------------------------------|
| context_id | ID of the cursor to be executed                       |
| pos        | Position of a dynamically defined column in the query |

- DBE\_SQL.DBE\_SQL\_GET\_RESULT\_LONG

This function returns the value of a long type (not LONG or BIGINT) in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

Different from DBE\_SQL.GET\_RESULT\_LONG, the length of the return value is not set and the entire BIGINT value is returned.

The prototype of the DBE\_SQL.DBE\_SQL\_GET\_RESULT\_LONG function is as follows:

```
DBE_SQL.DBE_SQL_GET_RESULT_LONG(
 context_id IN INTEGER,
 pos IN INTEGER
)
RETURN BIGINT;
```

**Table 10-258** DBE\_SQL.DBE\_SQL\_GET\_RESULT\_LONG parameters

| Parameter  | Description                                           |
|------------|-------------------------------------------------------|
| context_id | ID of the cursor to be executed                       |
| pos        | Position of a dynamically defined column in the query |

- DBE\_SQL.DBE\_SQL\_GET\_RESULT\_RAW

This function returns the value of the RAW type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

Different from DBE\_SQL.GET\_RESULT\_RAW, the length of the return value is not set and the entire string is returned.

The prototype of the DBE\_SQL.DBE\_SQL\_GET\_RESULT\_RAW function is as follows:

```
DBE_SQL.GET_RESULT_RAW(
 context_id IN INTEGER,
 pos IN INTEGER,
 tr INOUT RAW
)
RETURN RAW;
```

**Table 10-259** DBE\_SQL.GET\_RESULT\_RAW parameters

| Parameter  | Description                                           |
|------------|-------------------------------------------------------|
| context_id | ID of the cursor to be executed                       |
| pos        | Position of a dynamically defined column in the query |

- DBE\_SQL.IS\_ACTIVE

This function returns the status of a cursor. The status can be **open**, **parse**, **execute**, or **define**. If the status is **open**, the value is **TRUE**. If the status is unknown, an error is reported. In other cases, the value is **FALSE**.

The prototype of the DBE\_SQL.IS\_ACTIVE function is as follows:

```
DBE_SQL.IS_ACTIVE(
 context_id IN INTEGER
)
RETURN BOOLEAN;
```

**Table 10-260** DBE\_SQL.IS\_ACTIVE parameters

| Parameter  | Description                     |
|------------|---------------------------------|
| context_id | ID of the cursor to be queried. |

- DBE\_SQL.LAST\_ROW\_COUNT

Returns the accumulated count of data rows obtained after the latest NEXT\_ROW execution.

The prototype of the DBE\_SQL.LAST\_ROW\_COUNT function is as follows:

```
DBE_SQL.LAST_ROW_COUNT(
)
RETURN INTEGER;
```

- DBE\_SQL.RUN\_AND\_NEXT

Equivalent to calling SQL\_RUN and NEXT\_ROW in sequence.

The prototype of the DBE\_SQL.RUN\_AND\_NEXT function is as follows:

```
DBE_SQL.RUN_AND_NEXT(
 context_id IN INTEGER
)
RETURNS INTEGER;
```

**Table 10-261** DBE\_SQL.RUN\_AND\_NEXT parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor whose query statement is to be parsed |

- DBE\_SQL.SQL\_BIND\_VARIABLE

This function is used to bind a parameter to an SQL statement. When an SQL statement is executed, the SQL statement is executed based on the bound value.

The prototype of the DBE\_SQL.SQL\_BIND\_VARIABLE function is as follows:

```
DBE_SQL.SQL_BIND_VARIABLE(
 context_id IN int,
 query_string IN text,
 value IN anyelement,
 out_value_size IN int default null
)
RETURNS void;
```

**Table 10-262** DBE\_SQL.SQL\_BIND\_VARIABLE parameters

| Parameter      | Description                                            |
|----------------|--------------------------------------------------------|
| context_id     | ID of the cursor to be queried.                        |
| query_string   | Name of the bound variable                             |
| value          | Bound value                                            |
| out_value_size | Size of the return value. Default value: <b>null</b> . |

- DBE\_SQL.SQL\_BIND\_ARRAY

This function is used to bind a set of parameters to an SQL statement. When an SQL statement is executed, the SQL statement is executed based on the bound array.

The prototype of the DBE\_SQL.SQL\_BIND\_ARRAY function is as follows:

```
DBE_SQL.SQL_BIND_ARRAY(
 context_id IN int,
 query_string IN text,
 value IN anyarray
)
RETURNS void;
DBE_SQL.SQL_BIND_ARRAY(
 context_id IN int,
 query_string IN text,
 value IN anyarray,
 lower_index IN int,
 higher_index IN int
)
RETURNS void;
```

**Table 10-263** DBE\_SQL.SQL\_BIND\_ARRAY parameters

| Parameter    | Description                       |
|--------------|-----------------------------------|
| context_id   | ID of the cursor to be queried.   |
| query_string | Name of the bound variable.       |
| value        | Bound array.                      |
| lower_index  | Minimum index of the bound array. |
| higher_index | Maximum index of the bound array. |

- DBE\_SQL.SET\_RESULT\_TYPE\_INTS

This function defines columns of the INT array type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined

columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_INTS function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_INTS(
 context_id IN int,
 pos IN int,
 column_ref IN anyarray,
 cnt IN int,
 lower_bnd IN int
)
RETURNS integer;
```

**Table 10-264** DBE\_SQL.SET\_RESULT\_TYPE\_INTS parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be queried.                        |
| pos        | Position of a dynamically defined column in the query. |
| column_ref | Type of the returned array.                            |
| cnt        | Number of values obtained at a time.                   |
| lower_bnd  | Start index when an array is returned.                 |

- DBE\_SQL.SET\_RESULT\_TYPE\_TEXTS

This function defines columns of the TEXT array type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_TEXTS function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_TEXTS(
 context_id IN int,
 pos IN int,
 column_ref IN anyarray,
 cnt IN int,
 lower_bnd IN int,
 maxsize IN int
)
RETURNS integer;
```

**Table 10-265** DBE\_SQL.SET\_RESULT\_TYPE\_TEXTS parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be queried.                        |
| pos        | Position of a dynamically defined column in the query. |
| column_ref | Type of the returned array.                            |
| cnt        | Number of values obtained at a time.                   |
| lower_bnd  | Start index when an array is returned.                 |

| Parameter | Description                             |
|-----------|-----------------------------------------|
| maxsize   | Maximum length of the defined TEXT type |

- DBE\_SQL.SET\_RESULT\_TYPE\_RAWS

This function defines columns of the RAW array type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_RAWS function is as follows:

```
DBE_SQL.set_result_type_raws(
 context_id IN int,
 pos IN int,
 column_ref IN anyarray,
 cnt IN int,
 lower_bnd IN int,
 column_size IN int
)
RETURNS integer;
```

**Table 10-266** DBE\_SQL.SET\_RESULT\_TYPE\_RAWS parameters

| Parameter   | Description                                            |
|-------------|--------------------------------------------------------|
| context_id  | ID of the cursor to be queried.                        |
| pos         | Position of a dynamically defined column in the query. |
| column_ref  | Type of the returned array.                            |
| cnt         | Number of values obtained at a time.                   |
| lower_bnd   | Start index when an array is returned.                 |
| column_size | Column length                                          |

- DBE\_SQL.SET\_RESULT\_TYPE\_BYTEAS

This function defines columns of the BYTEA array type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_BYTEAS function is as follows:

```
DBE_SQL.set_result_type_byteas(
 context_id IN int,
 pos IN int,
 column_ref IN anyarray,
 cnt IN int,
 lower_bnd IN int,
 column_size IN int
)
RETURNS integer;
```

**Table 10-267** DBE\_SQL.SET\_RESULT\_TYPE\_BYTEAS parameters

| Parameter   | Description                                            |
|-------------|--------------------------------------------------------|
| context_id  | ID of the cursor to be queried.                        |
| pos         | Position of a dynamically defined column in the query. |
| column_ref  | Type of the returned array.                            |
| cnt         | Number of values obtained at a time.                   |
| lower_bnd   | Start index when an array is returned.                 |
| column_size | Column length                                          |

- **DBE\_SQL.SET\_RESULT\_TYPE\_CHARS**

This function defines columns of the CHAR array type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULT\_TYPE\_CHARS function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_CHARS(
 context_id IN int,
 pos IN int,
 column_ref IN anyarray,
 cnt IN int,
 lower_bnd IN int,
 column_size IN int
)
RETURNS integer;
```

**Table 10-268** DBE\_SQL.SET\_RESULT\_TYPE\_CHARS parameters

| Parameter   | Description                                            |
|-------------|--------------------------------------------------------|
| context_id  | ID of the cursor to be queried.                        |
| pos         | Position of a dynamically defined column in the query. |
| column_ref  | Type of the returned array.                            |
| cnt         | Number of values obtained at a time.                   |
| lower_bnd   | Start index when an array is returned.                 |
| column_size | Column length                                          |

- **DBE\_SQL.SET\_RESULTS\_TYPE**

This function defines columns returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE\_SQL.SET\_RESULTS\_TYPE function is as follows:

```
DBE_SQL.SET_RESULTS_TYPE(
 context_id IN int,
 pos IN int,
 column_ref IN anyarray,
 cnt IN int,
 lower_bnd IN int,
 maxsize IN int DEFAULT 1024
) returns void;
```

**Table 10-269** DBE\_SQL.SET\_RESULTS\_TYPE parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be queried.                        |
| pos        | Position of a dynamically defined column in the query. |
| column_ref | Type of the returned array.                            |
| cnt        | Number of values obtained at a time.                   |
| lower_bnd  | Start index when an array is returned.                 |
| maxsize    | Maximum length of the defined type                     |

- DBE\_SQL.GET\_RESULTS\_INT

This stored procedure returns the value of the INT array type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW. The prototype of the DBE\_SQL.GET\_RESULTS\_INT stored procedure is as follows:

```
DBE_SQL.GET_RESULTS_INT(
 context_id IN int,
 pos IN int,
 column_value INOUT anyarray
);
```

**Table 10-270** DBE\_SQL.GET\_RESULTS\_INT parameters

| Parameter    | Description                                            |
|--------------|--------------------------------------------------------|
| context_id   | ID of the cursor to be queried.                        |
| pos          | Position of a dynamically defined column in the query. |
| column_value | Return value                                           |

- DBE\_SQL.GET\_RESULTS\_TEXT

This stored procedure returns the value of the TEXT array type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW. The prototype of the DBE\_SQL.GET\_RESULTS\_TEXT stored procedure is as follows:

```
DBE_SQL.GET_RESULTS_TEXT(
 context_id IN int,
 pos IN int,
```

```
column_value INOUT anyarray
);
```

**Table 10-271** DBE\_SQL.GET\_RESULTS\_TEXT parameters

| Parameter    | Description                                            |
|--------------|--------------------------------------------------------|
| context_id   | ID of the cursor to be queried.                        |
| pos          | Position of a dynamically defined column in the query. |
| column_value | Return value                                           |

- DBE\_SQL.GET\_RESULTS\_RAW

This stored procedure returns the value of the RAW array type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

The prototype of the DBE\_SQL.GET\_RESULTS\_RAW stored procedure is as follows:

```
DBE_SQL.GET_RESULTS_RAW(
context_id IN int,
pos IN int,
column_value INOUT anyarray
);
```

**Table 10-272** DBE\_SQL.GET\_RESULTS\_RAW parameters

| Parameter    | Description                                            |
|--------------|--------------------------------------------------------|
| context_id   | ID of the cursor to be queried.                        |
| pos          | Position of a dynamically defined column in the query. |
| column_value | Return value                                           |

- DBE\_SQL.GET\_RESULTS\_BYTEA

This stored procedure returns the value of the BYTEA array type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

The prototype of the DBE\_SQL.GET\_RESULTS\_BYTEA stored procedure is as follows:

```
DBE_SQL.GET_RESULTS_BYTEA(
context_id IN int,
pos IN int,
column_value INOUT anyarray
);
```

**Table 10-273** DBE\_SQL.GET\_RESULTS\_BYTEA parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be queried.                        |
| pos        | Position of a dynamically defined column in the query. |

| Parameter    | Description  |
|--------------|--------------|
| column_value | Return value |

- DBE\_SQL.GET\_RESULTS\_CHAR

This stored procedure returns the value of the CHAR array type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

The prototype of the DBE\_SQL.GET\_RESULTS\_CHAR stored procedure is as follows:

```
DBE_SQL.GET_RESULTS_CHAR(
 context_id IN int,
 pos IN int,
 column_value INOUT anyarray
);
```

**Table 10-274** DBE\_SQL.GET\_RESULTS\_CHAR parameters

| Parameter    | Description                                            |
|--------------|--------------------------------------------------------|
| context_id   | ID of the cursor to be queried.                        |
| pos          | Position of a dynamically defined column in the query. |
| column_value | Return value                                           |

- DBE\_SQL.GET\_RESULTS

This stored procedure returns the value of the array type in a specified position of a cursor and accesses the data obtained by DBE\_SQL.NEXT\_ROW.

 **NOTE**

The bottom-layer mechanism of DBE\_SQL.GET\_RESULTS is implemented through arrays. When different arrays are used to obtain the return value of the same column, NULL values are filled in the array due to discontinuous internal indexes to ensure the continuity of array indexes. As a result, the length of the returned result array is different from that of Oracle Database.

The prototype of the DBE\_SQL.GET\_RESULTS stored procedure is as follows:

```
DBE_SQL.GET_RESULTS(
 context_id IN int,
 pos IN int,
 column_value INOUT anyarray
);
```

**Table 10-275** DBE\_SQL.GET\_RESULTS parameters

| Parameter    | Description                                            |
|--------------|--------------------------------------------------------|
| context_id   | ID of the cursor to be queried.                        |
| pos          | Position of a dynamically defined column in the query. |
| column_value | Return value                                           |

- **DBE\_SQL.SQL\_DESCRIBE\_COLUMNS**  
This function is used to describe column information and can be used only for cursors defined by SELECT.

The prototype of the DBE\_SQL.SQL\_DESCRIBE\_COLUMNS function is as follows:

```
DBE_SQL.SQL_DESCRIBE_COLUMNS(
 context_id IN int,
 col_cnt INOUT int,
 desc_t INOUT dbe_sql.desc_tab
)RETURNS record;
```

**Table 10-276** DBE\_SQL.SQL\_DESCRIBE\_COLUMNS parameters

| Parameter  | Description                        |
|------------|------------------------------------|
| context_id | ID of the cursor to be queried.    |
| col_cnt    | Number of columns returned         |
| desc_t     | Description of the returned column |

- **DBE\_SQL.DESCRIBE\_COLUMNS**  
Describes column information and can be used only for cursors defined by SELECT. This API is developed for compatibility purposes.

The prototype of the DBE\_SQL.DESCRIBE\_COLUMNS function is as follows:

```
DBE_SQL.DESCRIBE_COLUMNS(
 context_id IN int,
 col_cnt OUT int,
 desc_t OUT dbe_sql.desc_tab
)
```

**Table 10-277** DBE\_SQL.DESCRIBE\_COLUMNS parameters

| Parameter  | Description                        |
|------------|------------------------------------|
| context_id | ID of the cursor to be queried.    |
| col_cnt    | Number of columns returned         |
| desc_t     | Description of the returned column |

- **DBE\_SQL.BIND\_VARIABLE**  
This function is used to bind parameters. You are advised to use DBE\_SQL.SQL\_BIND\_VARIABLE.
- **DBE\_SQL.SQL\_SET\_RESULTS\_TYPE\_C**  
This function is used to dynamically define a column of the array type. You are advised not to use it.

The prototype of the DBE\_SQL.SQL\_SET\_RESULTS\_TYPE\_C function is as follows:

```
DBE_SQL.sql_set_results_type_c(
 context_id IN int,
 pos IN int,
 column_ref IN anyarray,
 cnt IN int,
```

```

lower_bnd IN int,
col_type IN anyelement,
maxsize IN int
)return integer;

```

**Table 10-278** DBE\_SQL.SQL\_SET\_RESULTS\_TYPE\_C parameters

| Parameter  | Description                                            |
|------------|--------------------------------------------------------|
| context_id | ID of the cursor to be queried.                        |
| pos        | Position of a dynamically defined column in the query. |
| column_ref | Type of the returned array.                            |
| cnt        | Number of values obtained at a time.                   |
| lower_bnd  | Start index when an array is returned.                 |
| col_type   | Variable type corresponding to the returned array type |
| maxsize    | Maximum length of the defined type                     |

- DBE\_SQL.SQL\_GET\_VALUES\_C

This function is used to read a dynamically defined column value. You are advised not to use it.

The prototype of the DBE\_SQL.SQL\_GET\_VALUES\_C function is as follows:

```

DBE_SQL.sql_get_values_c(
context_id IN int,
pos IN int,
results_type INOUT anyarray,
result_type IN anyelement
)return anyarray;

```

**Table 10-279** DBE\_SQL.SQL\_GET\_VALUES\_C parameters

| Parameter    | Description                     |
|--------------|---------------------------------|
| context_id   | ID of the cursor to be queried. |
| pos          | Parameter position              |
| results_type | Obtained result                 |
| result_type  | Type of the obtained result     |

- DBE\_SQL.GET\_VARIABLE\_RESULT

This stored procedure returns the value of the bound OUT parameter and obtains the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_VARIABLE\_RESULT stored procedure is as follows:

```

DBE_SQL.get_variable_result(
context_id IN int,
pos IN VARCHAR2,

```

```
column_value INOUT anyelement
);
```

**Table 10-280** DBE\_SQL.GET\_VARIABLE\_RESULT parameters

| Parameter    | Description                     |
|--------------|---------------------------------|
| context_id   | ID of the cursor to be queried. |
| pos          | Name of the bound parameter     |
| column_value | Return value                    |

- DBE\_SQL.GET\_VARIABLE\_RESULT\_CHAR

This function is used to return the value of the bound OUT parameter of the CHAR type and obtain the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_VARIABLE\_RESULT\_CHAR function is as follows:

```
DBE_SQL.get_variable_result_char(
 context_id IN int,
 pos IN VARCHAR2
)
RETURNS char
```

**Table 10-281** DBE\_SQL.GET\_VARIABLE\_RESULT\_CHAR parameters

| Parameter  | Description                     |
|------------|---------------------------------|
| context_id | ID of the cursor to be queried. |
| pos        | Name of the bound parameter     |

- DBE\_SQL.GET\_VARIABLE\_RESULT\_RAW

This stored procedure returns the value of the bound OUT parameter of the RAW type and obtains the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_VARIABLE\_RESULT\_RAW stored procedure is as follows:

```
CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_raw(
 context_id IN int,
 pos IN VARCHAR2,
 value INOUT anyelement
);
```

**Table 10-282** DBE\_SQL.GET\_VARIABLE\_RESULT\_RAW parameters

| Parameter  | Description                     |
|------------|---------------------------------|
| context_id | ID of the cursor to be queried. |
| pos        | Name of the bound parameter     |
| value      | Return value                    |

- DBE\_SQL.GET\_VARIABLE\_RESULT\_TEXT

This function is used to return the value of the bound OUT parameter of the TEXT type and obtain the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_VARIABLE\_RESULT\_TEXT function is as follows:

```
CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_text(
 context_id IN int,
 pos IN VARCHAR2
)
RETURNS text
```

**Table 10-283** DBE\_SQL.GET\_VARIABLE\_RESULT\_TEXT parameters

| Parameter  | Description                     |
|------------|---------------------------------|
| context_id | ID of the cursor to be queried. |
| pos        | Name of the bound parameter     |

- DBE\_SQL.GET\_VARIABLE\_RESULT\_INT

This stored procedure returns the value of the bound OUT parameter of the INT type and obtains the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_VARIABLE\_RESULT\_INT stored procedure is as follows:

```
DBE_SQL.get_variable_result_int(
 context_id IN int,
 pos IN VARCHAR2,
 value INOUT anyelement
);
```

**Table 10-284** DBE\_SQL.GET\_VARIABLE\_RESULT\_INT parameters

| Parameter  | Description                     |
|------------|---------------------------------|
| context_id | ID of the cursor to be queried. |
| pos        | Name of the bound parameter     |
| value      | Return value                    |

- DBE\_SQL.GET\_ARRAY\_RESULT\_TEXT

This stored procedure returns the value of the bound OUT parameter of the TEXT array type and obtains the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_ARRAY\_RESULT\_TEXT stored procedure is as follows:

```
DBE_SQL.get_array_result_text(
 context_id IN int,
 pos IN VARCHAR2,
 column_value INOUT anyarray
);
```

**Table 10-285** DBE\_SQL.GET\_ARRAY\_RESULT\_TEXT parameters

| Parameter  | Description                     |
|------------|---------------------------------|
| context_id | ID of the cursor to be queried. |

| Parameter    | Description                 |
|--------------|-----------------------------|
| pos          | Name of the bound parameter |
| column_value | Return value                |

- DBE\_SQL.GET\_ARRAY\_RESULT\_RAW

This stored procedure returns the value of the bound OUT parameter of the RAW array type and obtains the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_ARRAY\_RESULT\_RAW stored procedure is as follows:

```
DBE_SQL.get_array_result_raw(
 context_id IN int,
 pos IN VARCHAR2,
 column_value INOUT anyarray
);
```

**Table 10-286** DBE\_SQL.GET\_ARRAY\_RESULT\_RAW parameters

| Parameter    | Description                     |
|--------------|---------------------------------|
| context_id   | ID of the cursor to be queried. |
| pos          | Name of the bound parameter     |
| column_value | Return value                    |

- DBE\_SQL.GET\_ARRAY\_RESULT\_CHAR

This stored procedure returns the value of the bound OUT parameter of the CHAR array type and obtains the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_ARRAY\_RESULT\_CHAR stored procedure is as follows:

```
DBE_SQL.get_array_result_char(
 context_id IN int,
 pos IN VARCHAR2,
 column_value INOUT anyarray
);
```

**Table 10-287** DBE\_SQL.GET\_ARRAY\_RESULT\_CHAR parameters

| Parameter    | Description                     |
|--------------|---------------------------------|
| context_id   | ID of the cursor to be queried. |
| pos          | Name of the bound parameter     |
| column_value | Return value                    |

- DBE\_SQL.GET\_ARRAY\_RESULT\_INT

This stored procedure returns the value of the bound OUT parameter of the INT array type and obtains the OUT parameter in a stored procedure.

The prototype of the DBE\_SQL.GET\_ARRAY\_RESULT\_INT stored procedure is as follows:

```
DBE_SQL.get_array_result_int(
 context_id IN int,
 pos IN VARCHAR2,
 column_value INOUT anyarray
);
```

**Table 10-288** DBE\_SQL.GET\_ARRAY\_RESULT\_INT parameters

| Parameter    | Description                     |
|--------------|---------------------------------|
| context_id   | ID of the cursor to be queried. |
| pos          | Name of the bound parameter     |
| column_value | Return value                    |

## Examples

```
-- Example 1
-- Create a table and insert data into the table.
CREATE TABLE test_desc_cols(
 id NUMBER,
 name VARCHAR2(50)
);
INSERT INTO test_desc_cols(id, name) VALUES (1, 'xiaoming');
INSERT INTO test_desc_cols(id, name) VALUES (2, 'xiaohong');
INSERT INTO test_desc_cols(id, name) VALUES (3, 'xiaolan');

DECLARE
context_id INTEGER;
col_cnt INTEGER;
v_id int;
v_name varchar2;
execute_ret INTEGER;
BEGIN
-- Open a cursor.
context_id := DBE_SQL.REGISTER_CONTEXT();
-- Compile the cursor.
DBE_SQL.SQL_SET_SQL(context_id, 'SELECT * FROM test_desc_cols', 2);
-- Set the return value type of a column.
DBE_SQL.SET_RESULT_TYPE(context_id, 1, v_id);
DBE_SQL.SET_RESULT_TYPE(context_id, 2, v_name);
execute_ret := DBE_SQL.SQL_RUN(context_id);

loop
exit when (DBE_SQL.NEXT_ROW(context_id) <= 0);
-- Obtain values.
DBE_SQL.GET_RESULT(context_id, 1, v_id);
DBE_SQL.GET_RESULT(context_id, 2, v_name);
-- Output the result.
dbe_output.print_line('id: || v_id || ' name: ' || v_name);
end loop;

DBE_SQL.SQL_UNREGISTER_CONTEXT(context_id);
END;
/

CREATE OR REPLACE PROCEDURE test_square(n NUMBER, square OUT NUMBER) IS
BEGIN
 square := n * n;
END;
/

DECLARE
cur NUMBER;
```

```
query varchar(2000);
ret integer;
n NUMBER;
square Integer;
BEGIN
 n := 2;
 cur := DBE_SQL.REGISTER_CONTEXT();
 query := 'BEGIN test_square(:n_bnd, :square_bnd); END;';
 DBE_SQL.SQL_SET_SQL(cur, query, 2);
 DBE_SQL.SQL_BIND_VARIABLE(cur, 'n_bnd', n);
 DBE_SQL.SQL_BIND_VARIABLE(cur, 'square_bnd', square);
 ret := DBE_SQL.SQL_RUN(cur);
 DBE_SQL.GET_VARIABLE_RESULT(cur, 'square_bnd', square);
 DBE_OUTPUT.PRINT_LINE('square = ' || square);
 DBE_SQL.SQL_UNREGISTER_CONTEXT(cur);
END;
/

-- Example 2
-- Examples of executing DESCRIBE_COLUMNS, RUN_AND_NEXT, and LAST_ROW_COUNT
-- Create a stored procedure for printing column description information.
CREATE OR REPLACE PROCEDURE print_rec(
 rec in DBE_SQL.DESC_REC
)package AS
BEGIN
 raise INFO 'col_type = %', rec.col_type;
 raise INFO 'col_name = %', rec.col_name;
 raise INFO 'col_name_len = %', rec.col_name_len;
END;
/

-- Verify functions.
DECLARE
context_id INTEGER;
col_cnt INTEGER;
rec_tab DBE_SQL.DESC_TAB;
excute_ret INTEGER;
nextrow_ret INTEGER;
last_row_count INTEGER;
BEGIN
-- Open a cursor.
context_id := DBE_SQL.REGISTER_CONTEXT();
-- Compile the cursor.
DBE_SQL.SQL_SET_SQL(context_id, 'SELECT * FROM test_desc_cols', 2);
-- Print the column description information.
DBE_SQL.DESCRIBE_COLUMNS(context_id, col_cnt, rec_tab);
FOR var IN 1..col_cnt LOOP
 print_rec(rec_tab(var));
END LOOP;
-- Execute and obtain a row of data.
excute_ret := DBE_SQL.RUN_AND_NEXT(context_id);
-- Obtain a row of data.
nextrow_ret := DBE_SQL.NEXT_ROW(context_id);
-- Obtain the number of obtained data rows.
last_row_count := DBE_SQL.LAST_ROW_COUNT;
DBE_OUTPUT.PRINT_LINE('last_row_count = ' || last_row_count);
DBE_SQL.SQL_UNREGISTER_CONTEXT(context_id);
END;
/
```

### 10.11.2.8 DBE\_FILE

The DBE\_FILE package provides the capabilities of reading and writing OS text files for stored procedures.

## Precautions

- DBE\_FILE requires that files opened using DBE\_FILE.FOPEN be encoded using the database character set. If the opened files are not encoded using the expected character set, an encoding verification error occurs when DBE\_FILE.READ\_LINE is used to read files. DBE\_FILE requires that files opened using DBE\_FILE.FOPEN\_NCHAR be encoded using the UTF-8 character set. If the opened files are not encoded using the expected character set, an encoding verification error occurs when DBE\_FILE.READ\_LINE\_NCHAR is used to read files.
- When DBE\_OUTPUT.PUT\_LINE is used to print the result obtained by the DBE\_FILE.READ\_LINE\_NCHAR API, ensure that the UTF-8 character set encoding can be converted to the current database character set encoding. If the preceding conditions are met, the result can be properly output. DBE\_OUTPUT.PRINT\_LINE does not support this function.
- DBE\_FILE requires that the character set encoding of the client be the same as that of the database.
- Assume that the database character set encoding is ASCII, the client character set encoding supports Chinese, and the client calls DBE\_FILE.WRITE\_NCHAR or DBE\_FILE.WRITE\_LINE\_NCHAR to write Chinese content. If the entered content is in UTF-8 encoding format, the written content may not be encoded in UTF-8 format. An error may be reported when the DBE\_FILE.READ\_LINE\_NCHAR is used.

## Data Types

- DBE\_FILE.FILE\_TYPE  
Defines the representation of files in the DBE\_FILE package. The fields in DBE\_FILE.FILE\_TYPE are private fields of the DBE\_FILE package. Do not change the field value of the type defined in DBE\_FILE.FILE\_TYPE.

```
CREATE TYPE DBE_FILE.FILE_TYPE AS(
 id INTEGER,
 datatype INTEGER,
 byte_mode BOOLEAN
);
```

**Table 10-289** DBE\_FILE.FILE\_TYPE columns

| Parameter | Description                                                                                                                                                                  |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id        | File handle                                                                                                                                                                  |
| datatype  | File data type (CHAR, NCHAR, or binary). Currently, only CHAR and NCHAR files are supported. For a CHAR file, <b>1</b> is returned. For an NCHAR file, <b>2</b> is returned. |
| byte_mode | Indicates that the file is opened in binary mode ( <b>TRUE</b> ) or text mode ( <b>FALSE</b> ).                                                                              |

## APIs

**Table 10-290** lists all APIs supported by the **DBE\_FILE** package.

Table 10-290 DBE\_FILE

| API                                          | Description                                                                                                                                                                     |
|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_FILE.OPEN/DBE_FILE.FOPEN</a> | Opens a file based on the specified directory and file name. The corresponding file handle or the DBE_FILE.FILE_TYPE type object encapsulated with the file handle is returned. |
| <a href="#">DBE_FILE.IS_CLOSE</a>            | Checks whether a file is closed.                                                                                                                                                |
| <a href="#">DBE_FILE.IS_OPEN</a>             | Checks whether a file is opened.                                                                                                                                                |
| <a href="#">DBE_FILE.READ_LINE</a>           | Reads a line of a specified length from an open file handle.                                                                                                                    |
| <a href="#">DBE_FILE.WRITE</a>               | Writes data to an open file.                                                                                                                                                    |
| <a href="#">DBE_FILE.NEW_LINE</a>            | Writes one or more line terminators to an open file.                                                                                                                            |
| <a href="#">DBE_FILE.WRITE_LINE</a>          | Writes data to an open file and automatically appends a line terminator.                                                                                                        |
| <a href="#">DBE_FILE.FORMAT_WRITE</a>        | Writes data in a specified format to the buffer of an open file.                                                                                                                |
| <a href="#">DBE_FILE.GET_RAW</a>             | Reads RAW data from an open file.                                                                                                                                               |
| <a href="#">DBE_FILE.PUT_RAW</a>             | Writes RAW data to an open file.                                                                                                                                                |
| <a href="#">DBE_FILE.FLUSH</a>               | Writes cached data to a physical file.                                                                                                                                          |
| <a href="#">DBE_FILE.CLOSE</a>               | Closes an open file handle.                                                                                                                                                     |
| <a href="#">DBE_FILE.CLOSE_ALL</a>           | Closes all file handles opened in a session.                                                                                                                                    |
| <a href="#">DBE_FILE.REMOVE</a>              | Deletes a disk file. To perform this operation, you must have required permissions.                                                                                             |
| <a href="#">DBE_FILE.RENAME</a>              | Renames a disk file, which is similar to the mv command of Unix.                                                                                                                |
| <a href="#">DBE_FILE.COPY</a>                | Copies data in a continuous area to a new file. If <b>start_line</b> and <b>end_line</b> are omitted, the entire file is copied.                                                |
| <a href="#">DBE_FILE.GET_ATTR</a>            | Reads and returns the attributes of a disk file.                                                                                                                                |
| <a href="#">DBE_FILE.SEEK</a>                | Adjusts the position of a file pointer forward or backward based on the specified number of bytes.                                                                              |
| <a href="#">DBE_FILE.GET_POS</a>             | Returns the current offset of a file, in bytes.                                                                                                                                 |

| API                                         | Description                                                                                                                    |
|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_FILE.FOPEN_NCHAR</a>        | Opens a file of the NCHAR type based on the specified directory and file name.                                                 |
| <a href="#">DBE_FILE.WRITE_NCHAR</a>        | Writes data of the NVARCHAR2 type to the buffer of an open file of the NCHAR type.                                             |
| <a href="#">DBE_FILE.WRITE_LINE_NCHAR</a>   | Writes data of the NVARCHAR2 type to the buffer of an open file of the NCHAR type and automatically appends a line terminator. |
| <a href="#">DBE_FILE.FORMAT_WRITE_NCHAR</a> | Writes data of the NVARCHAR2 type to the buffer of an open file of the NCHAR type in a specified format.                       |
| <a href="#">DBE_FILE.READ_LINE_NCHAR</a>    | Reads a line of a specified length from an open file of the NCHAR type.                                                        |

- [DBE\\_FILE.OPEN/DBE\\_FILE.FOPEN](#)

Opens a file. You can specify the maximum line size. A maximum of 50 files can be opened in a session. This function returns a file handle of the INTEGER type. The function of [DBE\\_FILE.FOPEN](#) is similar to that of [DBE\\_FILE.OPEN](#). It returns an object of the type defined in [DBE\\_FILE.FILE\\_TYPE](#).

The prototypes of the [DBE\\_FILE.OPEN](#) and [DBE\\_FILE.FOPEN](#) functions are:

```
DBE_FILE.OPEN (
 dir IN TEXT,
 file_name IN TEXT,
 open_mode IN TEXT,
 max_line_size IN INTEGER DEFAULT 1024)
RETURN INTEGER;

DBE_FILE.FOPEN(
 dir IN TEXT,
 file_name IN TEXT,
 open_mode IN TEXT,
 max_line_size IN INTEGER DEFAULT 1024)
RETURN DBE_FILE.FILE_TYPE;
```

**Table 10-291** DBE\_FILE.OPEN/DBE\_FILE.FOPEN parameters

| Parameter     | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|---------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dir           | TEXT    | IN                     | No           | Directory of a file. It is a string, indicating an object name.<br><b>NOTE</b> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to read and write files in the file path specified by <b>safe_data_path</b>.</li> </ul> |
| file_name     | TEXT    | IN                     | No           | File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the <b>OPEN</b> function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).                                                                                                                                                                                                                                                                                                                         |
| open_mode     | TEXT    | IN                     | No           | Open mode of a specified file, including: <ul style="list-style-type: none"> <li><b>r</b>: read text</li> <li><b>w</b>: write text</li> <li><b>a</b>: append text</li> <li><b>rb</b>: read byte</li> <li><b>wb</b>: write byte</li> <li><b>ab</b>: append byte</li> </ul> <b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.                                                                                                                           |
| max_line_size | INTEGER | IN                     | Yes          | Maximum number of characters in each line, including newline characters. The minimum value is <b>1</b> and the maximum is <b>32767</b> . If this parameter is not specified, the default value <b>1024</b> is used.                                                                                                                                                                                                                                                                                                                                 |

- DBE\_FILE.IS\_CLOSE

Checks whether a file handle is closed. A Boolean value is returned. If an invalid file handle is detected, the **INVALID\_FILEHANDLE** exception is thrown.

The prototype of the DBE\_FILE.IS\_CLOSE function is as follows:

```
DBE_FILE.IS_CLOSE (
file IN INTEGER)
RETURN BOOLEAN;

DBE_FILE.IS_CLOSE(
file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```

**Table 10-292** DBE\_FILE.IS\_CLOSE parameters

| Parameter       | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                                             |
|-----------------|-------------------------------|------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------|
| file IN INTEGER | INTEGER or DBE_FILE.FILE_TYPE | IN                     | Yes          | File handle or DBE_FILE.FILE_TYPE object to be checked. If the value is empty, the DBE_FILE.IS_CLOSE API returns empty. |

- DBE\_FILE.IS\_OPEN

Checks whether a file handle is opened. A Boolean value is returned. If an invalid file handle is detected, the **INVALID\_FILEHANDLE** exception is thrown.

The prototype of the DBE\_FILE.IS\_OPEN function is as follows:

```
DBE_FILE.IS_OPEN(
file IN INTEGER)
RETURN BOOLEAN;

DBE_FILE.IS_OPEN(
file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```

**Table 10-293** DBE\_FILE.IS\_OPEN parameters

| Parameter | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                                                    |
|-----------|-------------------------------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER or DBE_FILE.FILE_TYPE | IN                     | Yes          | File handle or DBE_FILE.FILE_TYPE object to be checked. If the value is empty, the DBE_FILE.IS_OPEN API returns <b>FALSE</b> . |

- DBE\_FILE.READ\_LINE

Reads data from an open file and stores the read result to the buffer. It reads data to the end of each line excluding the line terminator, to the end of the file, or to the size specified by the **len** parameter. The length of the data to be read cannot exceed the value of **max\_line\_size** specified when the file is opened.

The prototype of the **DBE\_FILE.READ\_LINE** function is as follows:

```
DBE_FILE.READ_LINE (
 file IN INTEGER,
 buffer OUT VARCHAR2,
 len IN INTEGER DEFAULT NULL);
```

```
DBE_FILE.READ_LINE(
 file IN DBE_FILE.FILE_TYPE,
 buffer OUT TEXT,
 len IN INTEGER DEFAULT NULL);
```

**Table 10-294** DBE\_FILE.READ\_LINE parameters

| Parameter | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                           |
|-----------|-------------------------------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER or DBE_FILE_FILE_TYPE | IN                     | No           | File handle opened by calling the <b>OPEN</b> function. The file must be opened in read mode. Otherwise, the <b>INVALID_OPERATION</b> exception is thrown.            |
| buffer    | TEXT                          | OUTPUT                 | No           | Buffer used to receive data                                                                                                                                           |
| len       | INTEGER                       | IN                     | Yes          | Number of bytes read from a file. The default value is <b>NULL</b> . If the default value <b>NULL</b> is used, <b>max_line_size</b> is used to specify the line size. |

- **DBE\_FILE.WRITE**

Writes buffer data to the buffer corresponding to a file. The file must be opened in write mode. This operation does not write a line terminator.

The prototype of the **DBE\_FILE.WRITE** function is as follows:

```
DBE_FILE.WRITE(
 file IN INTEGER,
 buffer IN TEXT)
RETURN BOOLEAN;

DBE_FILE.WRITE(
 file IN DBE_FILE.FILE_TYPE,
 buffer IN TEXT)
RETURN VOID;
```

**Table 10-295** DBE\_FILE.WRITE parameters

| Parameter | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------|-------------------------------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER or DBE_FILE.FILE_TYPE | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN. The file must be opened in write mode. This operation does not write line terminators.                                                                                                                                                                                                                            |
| buffer    | TEXT                          | IN                     | Yes          | Text data to be written to the file. The accumulated write length of each line cannot be greater than or equal to the value of <b>max_line_size</b> specified when OPEN or FOPEN is used. Otherwise, an error is reported when the file is refreshed.<br><br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |

- **DBE\_FILE.NEW\_LINE**

Writes one or more line terminators to the buffer corresponding to a file. The line terminators are related to the platform used.

The prototype of the **DBE\_FILE.NEW\_LINE** function is as follows:

```
DBE_FILE.NEW_LINE(
 file IN INTEGER,
 line_nums IN INTEGER DEFAULT 1)
RETURN BOOLEAN;

DBE_FILE.NEW_LINE(
 file IN DBE_FILE.FILE_TYPE,
 line_nums IN INTEGER DEFAULT 1)
RETURN VOID;
```

**Table 10-296** DBE\_FILE.NEW\_LINE parameters

| Parameter | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                                                              |
|-----------|-------------------------------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER or DBE_FILE.FILE_TYPE | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.                                               |
| line_nums | INTEGER                       | IN                     | Yes          | Number of line terminators written to a file. The default value is 1. If this parameter is left blank, line terminators are not written. |

- **DBE\_FILE.WRITE\_LINE**

Writes buffer data to the buffer corresponding to a file. The file must be opened in write mode. This operation automatically adds a line terminator.

The prototype of the **DBE\_FILE.WRITE\_LINE** function is as follows:

```
DBE_FILE.WRITE_LINE(
 file IN INTEGER,
 buffer IN TEXT,
 flush IN BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;

DBE_FILE.WRITE_LINE(
 file IN DBE_FILE.FILE_TYPE,
 buffer IN TEXT,
 flush IN BOOLEAN DEFAULT FALSE)
RETURN VOID;
```

**Table 10-297** DBE\_FILE.WRITE\_LINE parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|---------|------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.                                                                                                                                                                                                                                                                                                                                                |
| buffer    | TEXT    | IN                     | Yes          | Text data to be written to the file. The length of each line (including the newline character) cannot be greater than the value of <b>max_line_size</b> specified when OPEN or FOPEN is executed or the default value. Otherwise, an error is reported when the file is refreshed.<br><br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |
| flush     | BOOLEAN | IN                     | Yes          | Specifies whether to flush data to disks after WRITE_LINE. The default value is <b>FALSE</b> .                                                                                                                                                                                                                                                                                                                                            |

- DBE\_FILE.FORMAT\_WRITE

Writes formatted data to the buffer corresponding to an open file. It is a DBE\_FILE.WRITE API that allows formatting.

The prototype of the **DBE\_FILE.FORMAT\_WRITE** function is as follows:

```
DBE_FILE.FORMAT_WRITE(
 file IN INTEGER,
 format IN TEXT,
 arg1 IN TEXT DEFAULT NULL,
 ...
 arg6 IN TEXT DEFAULT NULL)
RETURN BOOLEAN;

DBE_FILE.FORMAT_WRITE(
 file IN DBE_FILE.FILE_TYPE,
 format IN TEXT,
 arg1 IN TEXT DEFAULT NULL,
 ...
 arg6 IN TEXT DEFAULT NULL)
RETURN VOID;
```

**Table 10-298** DBE\_FILE.FORMAT\_WRITE parameters

| Parameter     | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                     |
|---------------|-------------------------------|------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file          | INTEGER or DBE_FILE.FILE_TYPE | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.                                                                                                                                                      |
| format        | TEXT                          | IN                     | Yes          | Formatted string, containing the text and format characters \n and %s. If this parameter is left blank, no data is written.                                                                                                                     |
| [arg1...arg6] | TEXT                          | IN                     | Yes          | Six optional parameters. The parameters and the positions of characters to be formatted are in one-to-one correspondence. If the parameter corresponding to a character to be formatted is not provided, an empty string is used to replace %s. |

- DBE\_FILE.GET\_RAW

Reads RAW data from an open file, stores the read result in the buffer, and returns the result from *r*.

The prototype of the DBE\_FILE.GET\_RAW stored procedure is as follows:

```
DBE_FILE.GET_RAW(
 file IN INTEGER,
 r OUT RAW,
 length IN INTEGER DEFAULT NULL);

DBE_FILE.GET_RAW(
 file IN DBE_FILE.FILE_TYPE,
 r OUT RAW,
 length IN INTEGER DEFAULT NULL);
```

**Table 10-299** DBE\_FILE.GET\_RAW parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                          |
|-----------|---------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.                                                                           |
| r         | RAW     | OUTPUT                 | No           | Buffer for receiving RAW data.                                                                                                                                       |
| length    | INTEGER | IN                     | Yes          | Number of bytes read from the file. The default value is <b>NULL</b> . If the value is <b>NULL</b> , the maximum length of the RAW type is used to specify the size. |

- **DBE\_FILE.PUT\_RAW**

Writes raw data to the buffer corresponding to a file.

The prototype of the DBE\_FILE.PUT\_RAW function is as follows:

```
DBE_FILE.PUT_RAW (
 file IN INTEGER,
 r IN RAW,
 flush IN BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;

DBE_FILE.PUT_RAW (
 file IN DBE_FILE.FILE_TYPE,
 r IN RAW,
 flush IN BOOLEAN DEFAULT FALSE)
RETURN VOID;
```

**Table 10-300** DBE\_FILE.PUT\_RAW parameters

| Parameter | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                          |
|-----------|-------------------------------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | INTEGER or DBE_FILE.FILE_TYPE | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.                                                                                           |
| r         | RAW                           | IN                     | No           | RAW data to be written to a file.<br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |
| flush     | BOOLEAN                       | IN                     | Yes          | Specifies whether to flush data to disks after PUT_RAW. The default value is <b>FALSE</b> .                                                                                          |

- DBE\_FILE.FLUSH

Writes cached data to a physical file. The cached data must have a line terminator.

The prototype of the DBE\_FILE.FLUSH function is as follows:

```
DBE_FILE.FLUSH(
 file IN INTEGER)
RETURN VOID;

DBE_FILE.FLUSH(
 file IN DBE_FILE.FILE_TYPE)
RETURN VOID;
```

**Table 10-301** DBE\_FILE.FLUSH parameters

| Parameter | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                |
|-----------|-------------------------------|------------------------|--------------|--------------------------------------------------------------------------------------------|
| file      | INTEGER or DBE_FILE.FILE_TYPE | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN. |

- DBE\_FILE.CLOSE

Closes an open file. When this function is called, if there is cached data to be written, exception information may be received and the return value is always **TRUE**.

The prototype of the DBE\_FILE.CLOSE function is as follows:

```
DBE_FILE.CLOSE(
 file IN INTEGER)
RETURN BOOLEAN;

DBE_FILE.CLOSE(
 file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```

**Table 10-302** DBE\_FILE.CLOSE parameters

| Parameter | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                |
|-----------|-------------------------------|------------------------|--------------|--------------------------------------------------------------------------------------------|
| file      | INTEGER or DBE_FILE.FILE_TYPE | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN. |

- **DBE\_FILE.CLOSE\_ALL**  
Closes all file handles opened in a session. This function can be used for emergency cleanup.

The prototype of the DBE\_FILE.CLOSE\_ALL function is as follows:

```
DBE_FILE.CLOSE_ALL()
RETRUN VOID;
```

- **DBE\_FILE.REMOVE**  
Deletes a disk file. To use this function, you must have the required permission.

The prototype of the DBE\_FILE.REMOVE function is as follows:

```
DBE_FILE.REMOVE(
 dir IN TEXT,
 file_name IN TEXT)
RETURN VOID;
```

**Table 10-303** DBE\_FILE.REMOVE parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------|------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dir       | TEXT | IN                     | No           | Directory of a file. It is a string, indicating an object name.<br><b>NOTE</b> <ul style="list-style-type: none"> <li>• Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>• When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to operate files in the file path specified by <b>safe_data_path</b>.</li> </ul> |
| file_name | TEXT | IN                     | No           | File name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

- **DBE\_FILE.RENAME**  
Renames a disk file. This function is similar to the mv command of Unix.

The prototype of the DBE\_FILE.RENAME function is as follows:

```
DBE_FILE.RENAME(
 src_dir IN TEXT,
 src_file_name IN TEXT,
 dest_dir IN TEXT,
 dest_file_name IN TEXT,
```

```

overwrite IN BOOLEAN DEFAULT FALSE)
RETURN VOID;

```

**Table 10-304** DBE\_FILE.RENAME parameters

| Parameter      | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------|---------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| src_dir        | TEXT    | IN                     | No           | Directory of the original file (case-sensitive)<br><b>NOTE</b> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to operate files in the file path specified by <b>safe_data_path</b>.</li> </ul> |
| src_file_name  | TEXT    | IN                     | No           | Original file to be renamed                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| dest_dir       | TEXT    | IN                     | No           | Target directory (case-sensitive).<br><b>NOTE</b> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to operate files in the file path specified by <b>safe_data_path</b>.</li> </ul>              |
| dest_file_name | TEXT    | IN                     | No           | New file name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| overwrite      | BOOLEAN | IN                     | Yes          | Specifies whether to overwrite the file. If the parameter is left empty or not specified, the parameter is not overwritten. If a file with the same name exists in the target directory, the file will not be overwritten.                                                                                                                                                                                                                                                                                                   |

- DBE\_FILE.COPY

Copies data in a continuous area to a new file. If **start\_line** and **end\_line** are omitted, the entire file is copied.

The prototype of the DBE\_FILE.COPY function is as follows:

```
DBE_FILE.COPY(
 src_dir IN TEXT,
 src_file_name IN TEXT,
 dest_dir IN TEXT,
 dest_file_name IN TEXT,
 start_line IN INTEGER DEFAULT 1,
 end_line IN INTEGER DEFAULT NULL)
RETURN VOID;
```

**Table 10-305** DBE\_FILE.COPY parameters

| Parameter     | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------|------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| src_dir       | TEXT | IN                     | No           | Directory of the original file<br><b>NOTE</b> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to operate files in the file path specified by <b>safe_data_path</b>.</li> </ul>    |
| src_file_name | TEXT | IN                     | No           | Name of the source file to be copied.                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| dest_dir      | TEXT | IN                     | No           | Directory of the destination file<br><b>NOTE</b> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to operate files in the file path specified by <b>safe_data_path</b>.</li> </ul> |

| Parameter      | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                    |
|----------------|---------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dest_file_name | TEXT    | IN                     | No           | Target file<br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |
| start_line     | INTEGER | IN                     | No           | Number of the line where the copy starts. The default value is <b>1</b> .                                                                                      |
| end_line       | INTEGER | IN                     | Yes          | Number of the line where the copy ends. The default value is <b>NULL</b> , indicating the end of the file.                                                     |

- DBE\_FILE.GET\_ATTR

Reads and returns the attributes of a disk file.

The prototype of the **DBE\_FILE.GET\_ATTR** function is as follows:

```
DBE_FILE.GET_ATTR(
 location IN TEXT,
 filename IN TEXT,
 fexists OUT BOOLEAN,
 file_length OUT BIGINT,
 block_size OUT INTEGER);
```

**Table 10-306** DBE\_FILE.GET\_ATTR parameters

| Parameter   | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------|---------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| location    | TEXT    | IN                     | No           | File directory<br><b>NOTE</b> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to operate files in the file path specified by <b>safe_data_path</b>.</li> </ul> |
| filename    | TEXT    | IN                     | No           | File name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| fileexists  | BOOLEAN | OUTPUT                 | No           | Specifies whether the file exists.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| file_length | BIGINT  | OUTPUT                 | No           | File length (unit: byte). If the file does not exist, <b>NULL</b> is returned.                                                                                                                                                                                                                                                                                                                                                                                                              |
| block_size  | INTEGER | OUTPUT                 | No           | Block size of the file system (unit: byte). If the file does not exist, <b>NULL</b> is returned.                                                                                                                                                                                                                                                                                                                                                                                            |

- **DBE\_FILE.SEEK**  
Adjusts the position of a file pointer forward or backward based on the specified number of bytes.

The prototype of the **DBE\_FILE.SEEK** function is as follows:

```
DBE_FILE.SEEK(
 file IN INTEGER,
 absolute_start IN BIGINT DEFAULT NULL,
 relative_start IN BIGINT DEFAULT NULL)
RETURN VOID;
```

```
DBE_FILE.SEEK(
file IN DBE_FILE.FILE_TYPE,
absolute_start IN BIGINT DEFAULT NULL,
relative_start IN BIGINT DEFAULT NULL)
RETURN VOID;
```

**Table 10-307** DBE\_FILE.SEEK parameters

| Parameter      | Type                          | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                            |
|----------------|-------------------------------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file           | INTEGER or DBE_FILE.FILE_TYPE | IN                     | No           | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.                                                                                                                                                                             |
| absolute_start | BIGINT                        | IN                     | Yes          | Absolute offset of a file. The default value is <b>NULL</b> .                                                                                                                                                                                                          |
| relative_start | BIGINT                        | IN                     | Yes          | Relative offset of a file. A positive number indicates forward offset and a negative number indicates backward offset. The default value is <b>NULL</b> . If both <b>absolute_start</b> and this parameter are specified, the <b>absolute_start</b> parameter is used. |

- DBE\_FILE.GET\_POS

Returns the current offset of the file in bytes.

The prototype of the DBE\_FILE.FGETPOS function is as follows:

```
DBE_FILE.GET_POS(
file IN INTEGER)
```

```
RETURN BIGINT;
DBE_FILE.GET_POS(
 file IN DBE_FILE.FILE_TYPE)
RETURN BIGINT;
```

**Table 10-308** DBE\_FILE.GET\_POS parameters

| Parameter | Type                                                                                       | In<br>pu<br>t/<br>Ou<br>tp<br>ut<br>Pa<br>ra<br>m<br>e<br>t<br>e<br>r | Ca<br>n<br>Be<br>Em<br>p<br>t<br>y | Description                                                                                |
|-----------|--------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|------------------------------------|--------------------------------------------------------------------------------------------|
| file      | IN<br>TE<br>GE<br>R<br>or<br>D<br>B<br>E<br>_<br>F<br>I<br>L<br>E<br>_<br>T<br>Y<br>P<br>E | IN                                                                    | No                                 | File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN. |

- **DBE\_FILE.FOPEN\_NCHAR**

Opens a file. You can specify the maximum line size. A maximum of 50 files can be opened in a session. This function returns a file handle of the DBE\_FILE.FILE\_TYPE type. This function opens a file in national character set mode for input or output.

The prototype of the DBE\_FILE.FOPEN\_NCHAR function is as follows:

```
DBE_FILE.FOPEN_NCHAR(
 dir IN TEXT,
 file_name IN TEXT,
 open_mode IN TEXT,
 max_line_size IN INTEGER DEFAULT 1024)
RETURN DBE_FILE.FILE_TYPE;
```

**Table 10-309** DBE\_FILE.FOPEN\_NCHAR parameters

| Parameter     | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|---------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dir           | TEXT    | IN                     | No           | Directory of a file. It is a string, indicating an object name.<br><b>NOTE</b> <ul style="list-style-type: none"> <li>Location of the file directory, which needs to be added to system catalog <b>PG_DIRECTORY</b>. If the input path does not match the path in <b>PG_DIRECTORY</b>, an error indicating that the path does not exist is reported.</li> <li>When the GUC parameter <b>safe_data_path</b> is enabled, you can only use an advanced package to read and write files in the file path specified by <b>safe_data_path</b>.</li> </ul> |
| file_name     | TEXT    | IN                     | No           | File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the <b>FOPEN_NCHAR</b> function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).                                                                                                                                                                                                                                                                                                                  |
| open_mode     | TEXT    | IN                     | No           | Open mode of a specified file, including: <ul style="list-style-type: none"> <li><b>r</b>: read text</li> <li><b>w</b>: write text</li> <li><b>a</b>: append text</li> <li><b>rb</b>: read byte</li> <li><b>wb</b>: write byte</li> <li><b>ab</b>: append byte</li> </ul> <b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.                                                                                                                           |
| max_line_size | INTEGER | IN                     | Yes          | Maximum number of characters in each line, including newline characters. The minimum value is <b>1</b> and the maximum is <b>32767</b> . If this parameter is not specified, the default value <b>1024</b> is used.                                                                                                                                                                                                                                                                                                                                 |

- DBE\_FILE.WRITE\_NCHAR

Writes data in the buffer to a file. The file must be opened in national character set or write mode. This operation does not write a line terminator, and the return value is always **TRUE**. The text string is written in the UTF8 character set format.

The prototype of the DBE\_FILE.WRITE\_NCHAR function is as follows:

```
DBE_FILE.WRITE_NCHAR(
 file IN DBE_FILE.FILE_TYPE,
 buffer IN NVARCHAR2)
RETURN VOID;
```

**Table 10-310** DBE\_FILE.WRITE\_NCHAR parameters

| Parameter | Type               | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------|--------------------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | DBE_FILE.FILE_TYPE | IN                     | No           | Object of the DBE_FILE.FILE_TYPE type opened using FOPEN_NCHAR. The file must be opened in write mode. This operation does not write line terminators.                                                                                                                                                                                                                                                   |
| buffer    | VARCHAR2           | IN                     | Yes          | Text data to be written to the file. The accumulated write length of each line cannot be greater than or equal to the value of <b>max_line_size</b> specified by <b>FOPEN_NCHAR</b> . Otherwise, an error is reported when the file is refreshed.<br><br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |

- DBE\_FILE.WRITE\_LINE\_NCHAR

Writes data in the buffer to a file. The file must be opened in national character set or write mode. This operation automatically appends line terminators, and the return value is always **TRUE**. The text string is written in the UTF8 character set format.

The prototype of the DBE\_FILE.WRITE\_LINE\_NCHAR function is as follows:

```
DBE_FILE.WRITE_LINE_NCHAR(
 file IN DBE_FILE.FILE_TYPE,
```

```
buffer IN NVARCHAR2)
RETURN VOID;
```

**Table 10-311** DBE\_FILE.WRITE\_LINE\_NCHAR parameters

| Parameter | Type               | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------|--------------------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | DBE_FILE.FILE_TYPE | IN                     | No           | Object of the DBE_FILE.FILE_TYPE type opened using FOPEN_NCHAR.                                                                                                                                                                                                                                                                                                                                                                  |
| buffer    | NVARCHAR2          | IN                     | Yes          | Text data to be written to the file. The length of each line (including the newline character) cannot be greater than the value of <b>max_line_size</b> specified by <b>FOPEN_NCHAR</b> or the default value. Otherwise, an error is reported when the file is refreshed.<br><br><b>NOTE</b><br>For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits. |

- DBE\_FILE.FORMAT\_WRITE\_NCHAR

Writes formatted data to an open file. It is a DBE\_FILE.WRITE\_NCHAR API that allows formatting. The return value is always **TRUE**.

The prototype of the DBE\_FILE.FORMAT\_WRITE\_NCHAR function is as follows:

```
DBE_FILE.FORMAT_WRITE_NCHAR(
 file IN DBE_FILE.FILE_TYPE,
 format IN NVARCHAR2,
 arg1 IN NVARCHAR2 DEFAULT NULL,
 ...
 arg5 IN NVARCHAR2 DEFAULT NULL)
RETURN VOID;
```

**Table 10-312** DBE\_FILE.FORMAT\_WRITE\_NCHAR parameters

| Parameter     | Type               | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                      |
|---------------|--------------------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file          | DBE_FILE.FILE_TYPE | IN                     | No           | Object of the DBE_FILE.FILE_TYPE type opened using FOPEN_NCHAR.                                                                                                                                                                                  |
| format        | VARCHAR2           | IN                     | Yes          | Formatted string, containing the text and format characters \n and %s.                                                                                                                                                                           |
| [arg1...arg5] | VARCHAR2           | IN                     | Yes          | Five optional parameters. The parameters and the positions of characters to be formatted are in one-to-one correspondence. If the parameter corresponding to a character to be formatted is not provided, an empty string is used to replace %s. |

- DBE\_FILE.READ\_LINE\_NCHAR

Reads data from an open file and stores the read result to the buffer. It reads data to the end of each line excluding the line terminator, to the end of the file, or to the size specified by the **len** parameter. The length of the data to be read cannot exceed the value of **max\_line\_size** specified by **FOPEN\_NCHAR**.

The prototype of the DBE\_FILE.READ\_LINE\_NCHAR stored procedure is as follows:

```
DBE_FILE.READ_LINE_NCHAR(
 file IN DBE_FILE.FILE_TYPE,
 buffer OUT NVARCHAR2,
 len IN INTEGER DEFAULT NULL);
```

**Table 10-313** DBE\_FILE.READ\_LINE\_NCHAR parameters

| Parameter | Type               | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                             |
|-----------|--------------------|------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file      | DBE_FILE.FILE_TYPE | IN                     | No           | Object of the DBE_FILE.FILE_TYPE type opened using FOPEN_NCHAR. The file must be opened in read mode. Otherwise, the INVALID_OPERATION exception is thrown.             |
| buffer    | VARCHAR2           | OUT                    | No           | Buffer for receiving data.                                                                                                                                              |
| len       | INTEGER            | IN                     | Yes          | Number of bytes read from the file. The default value is <b>NULL</b> . If the default value <b>NULL</b> is used, <b>max_line_size</b> is used to specify the line size. |

## Examples

```

-- Add the /temp/ directory to the PG_DIRECTORY system catalog as a system administrator.
CREATE OR REPLACE DIRECTORY dir AS '/tmp/';
-- The expected result is as follows:
CREATE DIRECTORY

-- Open a file and write data into the file.
DECLARE
 f integer;
 dir text := 'dir';
BEGIN
 f := dbe_file.open(dir, 'sample.txt', 'w');
 PERFORM dbe_file.write_line(f, 'ABC');
 PERFORM dbe_file.write_line(f, '123':numeric);
 PERFORM dbe_file.write_line(f, '----');
 PERFORM dbe_file.new_line(f);
 PERFORM dbe_file.write_line(f, '*****');
 PERFORM dbe_file.new_line(f, 0);
 PERFORM dbe_file.write_line(f, '+++++++');
 PERFORM dbe_file.new_line(f, 2);
 PERFORM dbe_file.write_line(f, '#####');
 PERFORM dbe_file.write(f, 'A');

```

```
PERFORM db_file.write(f, 'B');
PERFORM db_file.new_line(f);
PERFORM db_file.format_write(f, '[1 -> %s, 2 -> %s, 3 -> %s, 4 -> %s, 5 -> %s]', 'gaussdb', 'dbe', 'file',
'get', 'line');
PERFORM db_file.new_line(f);
PERFORM db_file.write_line(f, '1234567890');
f := db_file.close(f);
END;
/
-- The expected result is as follows:
ANONYMOUS BLOCK EXECUTE

-- Read data from the file mentioned above.
DECLARE
 f integer;
 dir text := 'dir';
BEGIN
 f := db_file.open(dir, 'sample.txt', 'r');
 FOR i IN 1..11 LOOP
 RAISE INFO '[%] : %', i, db_file.read_line(f);
 END LOOP;
END;
/
-- The expected result is as follows:
INFO: [1] : ABC
INFO: [2] : 123
INFO: [3] : -----
INFO: [4] :
INFO: [5] : *****
INFO: [6] : ++++++++
INFO: [7] :
INFO: [8] :
INFO: [9] : #####
INFO: [10] : AB
INFO: [11] : [1 -> gaussdb, 2 -> dbe, 3 -> file, 4 -> get, 5 -> line]
ANONYMOUS BLOCK EXECUTE

-- Offset the file handle and obtain the current file location.
DECLARE
 l_file integer;
 l_buffer VARCHAR2(32767);
 dir text := 'dir';
 abs_offset number := 100;
 rel_offset number := NULL;
BEGIN
 l_file := db_file.open(dir => dir, file_name => 'sample.txt', open_mode => 'R');
 db_output.print_line('before seek: current position is ' || db_file.get_pos(file => l_file)); -- before seek:
current position is 0
 db_file.seek(file => l_file, absolute_start=>abs_offset, relative_start=>rel_offset);
 db_output.print_line('fseek: current position is ' || db_file.get_pos(file => l_file)); -- seek: current
position is 100
 l_file := db_file.close(file => l_file);
END;
/
-- The expected result is as follows:
before seek: current position is 0
fseek: current position is 100
ANONYMOUS BLOCK EXECUTE

-- NCHAR read/write API cases
DECLARE
 f DBE_FILE.FILE_TYPE;
 buffer NVARCHAR2;
BEGIN
 -- Write a file.
 f := DBE_FILE.FOPEN_NCHAR('dir', 'sample02.txt', 'w');
 DBE_FILE.WRITE_NCHAR(f, 'A');
 DBE_FILE.WRITE_NCHAR(f, 'B');
 DBE_FILE.WRITE_NCHAR(f, 'C');
```

```

DBE_FILE.NEW_LINE(f);
DBE_FILE.WRITE_LINE_NCHAR(f, 'ABC');
DBE_FILE.FORMAT_WRITE_NCHAR(f, '[1 -> %s, 2 -> %s]\n', 'GaussDB', 'DBE_FILE');
DBE_FILE.CLOSE(f);
-- Read a file.
f := DBE_FILE.FOPEN_NCHAR('dir', 'sample02.txt', 'r');
DBE_FILE.READ_LINE_NCHAR(f, buffer); -- ABC
DBE_FILE.READ_LINE_NCHAR(f, buffer); -- ABC
DBE_FILE.READ_LINE_NCHAR(f, buffer); -- [1 -> GaussDB, 2 -> DBE_FILE]
DBE_OUTPUT.PRINT_LINE(buffer);
DBE_FILE.CLOSE(f);
END;
/
-- The expected result is as follows:
[1 -> GaussDB, 2 -> DBE_FILE]
ANONYMOUS BLOCK EXECUTE

```

### 10.11.2.9 DBE\_SESSION

#### APIs

**Table 10-314** provides all APIs supported by the **DBE\_SESSION** package. **DBE\_SESSION** takes effect at the session level.

**Table 10-314** DBE\_SESSION

| API                               | Description                                               |
|-----------------------------------|-----------------------------------------------------------|
| <b>DBE_SESSION.SET_CONTEXT</b>    | Sets the value of an attribute in a specified context.    |
| <b>DBE_SESSION.CLEAR_CONTEXT</b>  | Clears the value of an attribute in a specified context.  |
| <b>DBE_SESSION.SEARCH_CONTEXT</b> | Queries the value of an attribute in a specified context. |

- **DBE\_SESSION.SET\_CONTEXT**

Sets the value of an attribute in a specified namespace (context). The **DBE\_SESSION.SET\_CONTEXT** function prototype is as follows:

```

DBE_SESSION.SET_CONTEXT(
 namespace text,
 attribute text,
 value text
)returns void;

```

**Table 10-315** DBE\_SESSION.SET\_CONTEXT API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                      |
|-----------|------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| namespace | TEXT | IN                     | No           | Name of the context to be set. If the context does not exist, create a context. The value contains a maximum of 128 bytes. If the value exceeds 128 bytes, it will be truncated. |
| attribute | TEXT | IN                     | No           | Attribute name. The value contains a maximum of 1024 bytes. If the value exceeds 1024 bytes, it will be truncated.                                                               |
| value     | TEXT | IN                     | No           | Name of the value to be set. The value contains a maximum of 1024 bytes. If the value exceeds 1024 bytes, it will be truncated.                                                  |

- DBE\_SESSION.CLEAR\_CONTEXT

Clears the value of an attribute in a specified namespace (context). The **DBE\_SESSION.CLEAR\_CONTEXT** function prototype is as follows:

```
DBE_SESSION.CLEAR_CONTEXT (
 namespace text,
 client_identifier text default null,
 attribute text default null
)returns void ;
```

**Table 10-316** DBE\_SESSION.CLEAR\_CONTEXT API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                     |
|-----------|------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------|
| namespace | TEXT | IN                     | No           | Context specified by the user. The value contains a maximum of 128 bytes. If the value exceeds 128 bytes, it will be truncated. |

| Parameter             | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------|------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| client_iden<br>tifier | TEXT | IN                     | Yes          | Client authentication. The default value is <b>null</b> . Generally, you do not need to manually set this parameter.                                                                                                                                                                                                                                                             |
| attribute             | TEXT | IN                     | Yes          | Attribute to be cleared. The default value is <b>null</b> , indicating that all attributes of the specified context are cleared. The value contains a maximum of 1024 bytes. If the value exceeds 1024 bytes, it will be truncated.<br><b>CAUTION</b><br>To ensure forward compatibility, if the parameter value is 'null', all attributes of the specified context are cleared. |

- DBE\_SESSION.SEARCH\_CONTEXT

Queries the value of an attribute in a specified namespace (context). The **DBE\_SESSION.SEARCH\_CONTEXT** function prototype is:

```
DBE_SESSION.SEARCH_CONTEXT (
 namespace text,
 attribute text
)returns text;
```

**Table 10-317** DBE\_SESSION.SEARCH\_CONTEXT API parameters

| Parameter     | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                      |
|---------------|------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------|
| namespa<br>ce | TEXT | IN                     | No           | Context specified by the user. The value contains a maximum of 128 bytes. If the value exceeds 128 bytes, it will be truncated.  |
| attribute     | TEXT | IN                     | No           | Attribute to be searched for. The value contains a maximum of 1024 bytes. If the value exceeds 1024 bytes, it will be truncated. |

## Example

```

DECLARE
 a text;
BEGIN
 DBE_SESSION.set_context('test', 'gaussdb', 'one'); --Set the gaussdb attribute in the test context to one.
 a := DBE_SESSION.search_context('test', 'gaussdb');
 DBE_OUTPUT.PRINT_LINE(a);
 DBE_SESSION.clear_context('test', 'test', 'gaussdb');
END;
/
-- The expected result is as follows:
one
ANONYMOUS BLOCK EXECUTE

```

### 10.11.2.10 DBE\_MATCH

#### Interface Description

[Table 10-318](#) provides all interfaces supported by the **DBE\_MATCH** package.

**Table 10-318** DBE\_MATCH

| Interface                                          | Description                                                                                                                                                                                                                                                                                                                            |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DBE_MATCH.EDIT_DISTANCE_SIMILARITY</a> | Compares the difference between two character strings (minimum steps of deletion, addition, and conversion) and normalizes the difference to a value ranging from 0 to 100. The value <b>100</b> indicates that the two character strings are the same, and the value <b>0</b> indicates that the two character strings are different. |

- [DBE\\_MATCH.EDIT\\_DISTANCE\\_SIMILARITY](#)

Compares the difference between two character strings (minimum steps of deletion, addition, and conversion) and normalizes the difference to a value ranging from 0 to 100. The value **100** indicates that the two character strings are the same, and the value **0** indicates that the two character strings are different. The **DBE\_MATCH.EDIT\_DISTANCE\_SIMILARITY** function prototype is as follows:

```

DBE_MATCH.EDIT_DISTANCE_SIMILARITY(
 str1 IN text,
 str2 IN text
)returns integer ;

```

**Table 10-319** DBE\_MATCH.EDIT\_DISTANCE\_SIMILARITY interface parameters

| Parameter | Description                                                                  |
|-----------|------------------------------------------------------------------------------|
| str1      | First character string. If the value is <b>null</b> , <b>0</b> is returned.  |
| str2      | Second character string. If the value is <b>null</b> , <b>0</b> is returned. |

## 10.11.2.11 DBE\_SCHEDULER

### APIs

The advanced package **DBE\_SCHEDULER** supports more flexible creation of scheduled jobs through scheduling and programming. For details about all the supported APIs, see [Table 10-320](#).

#### NOTICE

DBE\_SCHEDULER does not support scheduled jobs for synchronizing data between nodes. To create scheduled jobs for multiple nodes, use [DBE\\_TASK](#).

**Table 10-320** DBE\_SCHEDULER

| API                                                   | Description                            |
|-------------------------------------------------------|----------------------------------------|
| <a href="#">DBE_SCHEDULER.CREATE_JOB</a>              | Creates a scheduled job.               |
| <a href="#">DBE_SCHEDULER.DROP_JOB</a>                | Deletes a scheduled job.               |
| <a href="#">DBE_SCHEDULER.DROP_SINGLE_JOB</a>         | Deletes a single scheduled job.        |
| <a href="#">DBE_SCHEDULER.SET_ATTRIBUTE</a>           | Sets object attributes.                |
| <a href="#">DBE_SCHEDULER.RUN_JOB</a>                 | Executes a scheduled job.              |
| <a href="#">DBE_SCHEDULER.RUN_BACKEND_JOB</a>         | Runs a scheduled job in the backend.   |
| <a href="#">DBE_SCHEDULER.RUN_FOREGROUND_JOB</a>      | Runs a scheduled job in the frontend.  |
| <a href="#">DBE_SCHEDULER.STOP_JOB</a>                | Stops a scheduled job.                 |
| <a href="#">DBE_SCHEDULER.STOP_SINGLE_JOB</a>         | Stops a single scheduled job.          |
| <a href="#">DBE_SCHEDULER.GENERATE_JOB_NAME</a>       | Generates the name of a scheduled job. |
| <a href="#">DBE_SCHEDULER.CREATE_PROGRAM</a>          | Creates a program.                     |
| <a href="#">DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT</a> | Defines program parameters.            |

| API                                            | Description                                        |
|------------------------------------------------|----------------------------------------------------|
| <b>DBE_SCHEDULER.DROP_PROGRAM</b>              | Deletes a program.                                 |
| <b>DBE_SCHEDULER.DROP_SINGLE_PROGRAM</b>       | Deletes a single program.                          |
| <b>DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE</b>    | Sets the parameters of a scheduled job.            |
| <b>DBE_SCHEDULER.CREATE_SCHEDULE</b>           | Creates a schedule.                                |
| <b>DBE_SCHEDULER.DROP_SCHEDULE</b>             | Deletes a schedule.                                |
| <b>DBE_SCHEDULER.DROP_SINGLE_SCHEDULE</b>      | Deletes a single schedule.                         |
| <b>DBE_SCHEDULER.CREATE_JOB_CLASS</b>          | Creates the class of a scheduled job.              |
| <b>DBE_SCHEDULER.DROP_JOB_CLASS</b>            | Deletes the class of a scheduled job.              |
| <b>DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS</b>     | Deletes the class of a single scheduled job.       |
| <b>DBE_SCHEDULER.GRANT_USER_AUTHORIZATION</b>  | Grants special permissions to a user.              |
| <b>DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION</b> | Revokes special permissions from a user.           |
| <b>DBE_SCHEDULER.CREATE_CREDENTIAL</b>         | Creates a certificate.                             |
| <b>DBE_SCHEDULER.DROP_CREDENTIAL</b>           | Destroys a certificate.                            |
| <b>DBE_SCHEDULER.ENABLE</b>                    | Enables an object.                                 |
| <b>DBE_SCHEDULER.ENABLE_SINGLE</b>             | Enables a single object.                           |
| <b>DBE_SCHEDULER.DISABLE</b>                   | Disables an object.                                |
| <b>DBE_SCHEDULER.DISABLE_SINGLE</b>            | Disables a single object.                          |
| <b>DBE_SCHEDULER.EVAL_CALENDAR_STRING</b>      | Analyzes character strings in the Calendar format. |
| <b>DBE_SCHEDULER.EVALUATE_CALENDAR_STRING</b>  | Analyzes character strings in the Calendar format. |

- DBE\_SCHEDULER.CREATE\_JOB

Creates a scheduled job.

The prototypes of the DBE\_SCHEDULER.CREATE\_JOB function are as follows:

```
-- Scheduled jobs of an inline schedule and a program.
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
job_type TEXT,
job_action TEXT,
number_of_arguments INTEGER DEFAULT 0,
start_date TIMESTAMP WITH TIME ZONE DEFAULT NULL,
repeat_interval TEXT DEFAULT NULL,
end_date TIMESTAMP WITH TIME ZONE DEFAULT NULL,
job_class TEXT DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN DEFAULT FALSE,
auto_drop BOOLEAN DEFAULT TRUE,
comments TEXT DEFAULT NULL,
credential_name TEXT DEFAULT NULL,
destination_name TEXT DEFAULT NULL
)

-- Reference the created scheduled jobs of the schedule and the program.
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
program_name TEXT,
schedule_name TEXT,
job_class TEXT DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN DEFAULT FALSE,
auto_drop BOOLEAN DEFAULT TRUE,
comments TEXT DEFAULT NULL,
job_style TEXT DEFAULT 'REGULAR',
credential_name TEXT DEFAULT NULL,
destination_name TEXT DEFAULT NULL
)

-- Reference the created program and the scheduled job of the inline schedule.
DBE_SCHEDULER.CREATE_JOB(
job_name text,
program_name TEXT,
start_date TIMESTAMP WITH TIME ZONE DEFAULT NULL,
repeat_interval TEXT DEFAULT NULL,
end_date TIMESTAMP WITH TIME ZONE DEFAULT NULL,
job_class TEXT DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN DEFAULT FALSE,
auto_drop BOOLEAN DEFAULT TRUE,
comments TEXT DEFAULT NULL,
job_style TEXT DEFAULT 'REGULAR',
credential_name TEXT DEFAULT NULL,
destination_name TEXT DEFAULT NULL
)

-- Reference the created schedule and the scheduled job of the inline program.
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
schedule_name TEXT,
job_type TEXT,
job_action TEXT,
number_of_arguments INTEGER DEFAULT 0,
job_class TEXT DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN DEFAULT FALSE,
auto_drop BOOLEAN DEFAULT TRUE,
comments TEXT DEFAULT NULL,
credential_name TEXT DEFAULT NULL,
destination_name TEXT DEFAULT NULL
)
```

 **NOTE**

The scheduled job created through **DBE\_SCHEDULER** does not conflict with the scheduled job in **DBE\_TASK**.

The scheduled job created by **DBE\_SCHEDULER** generates the corresponding **job\_id**. However, **job\_id** is meaningless.

**Table 10-321** DBE\_SCHEDULER.CREATE\_JOB API parameters

| Parameter           | Type                     | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                |
|---------------------|--------------------------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| job_name            | text                     | IN                     | No           | Name of a scheduled job.                                                                                                                                                                                                                                                                                   |
| job_type            | text                     | IN                     | No           | Inline program type of a scheduled job. The options are as follows: <ul style="list-style-type: none"> <li>• <b>'PLSQL_BLOCK'</b>: fast anonymous stored procedure.</li> <li>• <b>'STORED_PROCEDURE'</b>: stored procedure that is saved.</li> <li>• <b>'EXTERNAL_SCRIPT'</b>: external script.</li> </ul> |
| job_action          | text                     | IN                     | No           | Content executed by an inline program of a scheduled job.                                                                                                                                                                                                                                                  |
| number_of_arguments | integer                  | IN                     | No           | Number of inline program parameters of a scheduled job.                                                                                                                                                                                                                                                    |
| program_name        | text                     | IN                     | No           | Name of the program referenced by a scheduled job.                                                                                                                                                                                                                                                         |
| start_date          | timestamp with time zone | IN                     | Yes          | Inline scheduling start time of a scheduled job.                                                                                                                                                                                                                                                           |
| repeat_interval     | text                     | IN                     | Yes          | Inline scheduling period of a scheduled job.                                                                                                                                                                                                                                                               |
| end_date            | timestamp with time zone | IN                     | Yes          | Inline scheduling expiration time of a scheduled job.                                                                                                                                                                                                                                                      |

| Parameter        | Type    | Input/Output Parameter | Can Be Empty | Description                                                              |
|------------------|---------|------------------------|--------------|--------------------------------------------------------------------------|
| schedule_name    | text    | IN                     | No           | Name of the schedule referenced by a scheduled job.                      |
| job_class        | text    | IN                     | No           | Class name of a scheduled job.                                           |
| enabled          | boolean | IN                     | No           | Status of a scheduled job.                                               |
| auto_drop        | boolean | IN                     | No           | Automatic deletion of a scheduled job.                                   |
| comments         | text    | IN                     | Yes          | Comments.                                                                |
| job_style        | text    | IN                     | No           | Behavior pattern of a scheduled job. Only <b>'REGULAR'</b> is supported. |
| credential_name  | text    | IN                     | Yes          | Certificate name of a scheduled job.                                     |
| destination_name | text    | IN                     | Yes          | Target name of a scheduled job.                                          |

**Example:**

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 3, false, 'test');
create_program
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
create_schedule
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
create_job
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
```

```
drop_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program

(1 row)
```

### NOTICE

To create a scheduled job of the EXTERNAL\_SCRIPT type, the administrator must assign related permissions and certificates and the user who starts the database must have the read permission on the external script.

- DBE\_SCHEDULER.DROP\_JOB

Deletes a scheduled job.

The prototype of the DBE\_SCHEDULER.DROP\_JOB function is as follows:

```
DBE_SCHEDULER.drop_job(
job_name text,
force boolean default false,
defer boolean default false,
commit_semantics text default 'STOP_ON_FIRST_ERROR'
)
```

### NOTE

You can specify one or more jobs, or specify a job class in DBE\_SCHEDULER.DROP\_JOB to delete scheduled jobs.

**Table 10-322** DBE\_SCHEDULER.DROP\_JOB API parameters

| Parameter | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                  |
|-----------|---------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| job_name  | text    | IN                     | No           | Name or class of a scheduled job. You can specify one or more scheduled jobs. If you specify multiple scheduled jobs, separate them with commas (,).                                         |
| force     | boolean | IN                     | No           | Specifies whether to delete a scheduled job.<br><b>true:</b> The current scheduled job is stopped and then deleted.<br><b>false:</b> The scheduled job fails to be deleted if it is running. |

| Parameter        | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------|---------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| defer            | boolean | IN                     | No           | Specifies whether to delete a scheduled job.<br><b>true</b> : A scheduled job can be deleted after it is complete.                                                                                                                                                                                                                                                                                 |
| commit_semantics | text    | IN                     | No           | Commit rules:<br><b>'STOP_ON_FIRST_ERROR'</b> : The deletion operation performed before the first error is reported is committed.<br><b>'TRANSACTIONAL'</b> : Transaction-level commit. The deletion operation performed before an error is reported will be rolled back.<br><b>'ABSORB_ERRORS'</b> : Attempt to bypass an error and commit the deletion operation that is performed successfully. |

**Example:**

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 3, false, 'test');
create_program
```

-----  
(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
create_schedule
```

-----  
(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
create_job
```

-----  
(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
```

-----  
(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
drop_schedule
```

```

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program

(1 row)

```

### NOTICE

The **TRANSACTIONAL** option in **commit\_semantic** takes effect only when **force** is set to **false**.

- **DBE\_SCHEDULER.DROP\_SINGLE\_JOB**

Deletes a scheduled job.

The prototype of the **DBE\_SCHEDULER.DROP\_SINGLE\_JOB** function is as follows:

```

DBE_SCHEDULER.drop_single_job(
job_name text,
force boolean default false,
defer boolean default false
)

```

Example:

```

gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 0, false, 'test');
create_program

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_job('job1', 'program1', '2021-07-20', 'interval "3 minute"',
'2121-07-20', 'DEFAULT_JOB_CLASS', false, false, 'test', 'style', NULL, NULL);
create_job

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_job('job1', false, false);
drop_single_job

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program

(1 row)

```

- **DBE\_SCHEDULER.SET\_ATTRIBUTE**

Modifies the attributes of a scheduled job.

The prototypes of the **DBE\_SCHEDULER.SET\_ATTRIBUTE** function are as follows:

```

DBE_SCHEDULER.set_attribute(
name text,
attribute text,
value boolean
)

```

```

DBE_SCHEDULER.set_attribute(
name text,
attribute text,
value text
)

DBE_SCHEDULER.set_attribute(
name text,
attribute text,
value timestamp
)

DBE_SCHEDULER.set_attribute(
name text,
attribute text,
value timestamp with time zone
)

DBE_SCHEDULER.set_attribute(
name text,
attribute text,
value text,
value2 text default NULL
)

```

 **NOTE**

**name** specifies any object in **DBE\_SCHEDULER**.

**Table 10-323** DBE\_SCHEDULER.SET\_ATTRIBUTE API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description     |
|-----------|------|------------------------|--------------|-----------------|
| name      | text | IN                     | No           | Object name.    |
| attribute | text | IN                     | No           | Attribute name. |

| Parameter | Type                                                 | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------|------------------------------------------------------|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| value     | boolean/date/timestamp/timestamp with time zone/text | IN                     | No           | Attribute value. The options are as follows: <ul style="list-style-type: none"> <li>• Scheduled-job-related: job_type, job_action, number_of_arguments, start_date, repeat_interval, end_date, job_class, enabled, auto_drop, comments, credential_name, destination_name, program_name, schedule_name, and job_style.</li> <li>• Program-related: program_action, program_type, number_of_arguments, and comments.</li> <li>• Scheduling-related: start_date, repeat_interval, end_date, and comments.</li> </ul> |
| value2    | text                                                 | IN                     | Yes          | Additional attribute value. Reserved parameter bit. Currently, the target attribute with extra attribute values is not supported.                                                                                                                                                                                                                                                                                                                                                                                  |

**Example:**

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 3, false, 'test');
create_program
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.set_attribute('program1', 'number_of_arguments', 0);
set_attribute
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.set_attribute('program1', 'program_type', 'STORED_PROCEDURE');
set_attribute
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program
```

-----

(1 row)

**NOTICE**

Do not use DBE\_SCHEDULER.SET\_ATTRIBUTE to leave the parameters empty. The object name cannot be changed using DBE\_SCHEDULER.SET\_ATTRIBUTE. Inline objects cannot be changed by DBE\_SCHEDULER.SET\_ATTRIBUTE.

- **DBE\_SCHEDULER.RUN\_JOB**

Executes a scheduled job.

The prototype of the DBE\_SCHEDULER.RUN\_JOB function is as follows:

```
DBE_SCHEDULER.run_job(
job_name text,
use_current_session boolean default true
)
```

 **NOTE**

DBE\_SCHEDULER.RUN\_JOB is used to run scheduled jobs immediately. It is independent of the scheduling of scheduled jobs and can even run at the same time.

**Table 10-324** DBE\_SCHEDULER.RUN\_JOB API parameters

| Parameter           | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                        |
|---------------------|---------|------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| job_name            | text    | IN                     | No           | Name of a scheduled job. You can specify one or more scheduled jobs. If you specify multiple scheduled jobs, separate them with commas (,).                                                                                                                                                        |
| use_current_session | boolean | IN                     | No           | Specifies whether to run a scheduled job. <ul style="list-style-type: none"> <li>• <b>true:</b> Use the current session to check whether the scheduled job can run properly.</li> <li>• <b>false:</b> Start the scheduled job in the backend. The execution result is recorded in logs.</li> </ul> |

Example:

```
gaussdb=# SELECT db_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;','0,null,null,null','DEFAULT_JOB_CLASS',false,false,null,null,null);
```

```

create_job

(1 row)

gaussdb=# CALL DBE_SCHEDULER.run_job('job1', false);
run_job

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job

(1 row)

```

### NOTICE

Currently, **use\_current\_session** applies only to scheduled jobs whose **job\_type** is set to **EXTERNAL\_SCRIPT**.

- **DBE\_SCHEDULER.RUN\_BACKEND\_JOB**

Runs a scheduled job in the backend.

The prototype of the DBE\_SCHEDULER.RUN\_BACKEND\_JOB function is as follows:

```

DBE_SCHEDULER.run_backend_job(
job_name text
)

```

Example:

```

gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
create_job

(1 row)

gaussdb=# CALL DBE_SCHEDULER.run_backend_job('job1');
run_backend_job

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job

(1 row)

```

- **DBE\_SCHEDULER.RUN\_FOREGROUND\_JOB**

Executes a scheduled job in the current session.

Only external jobs can be executed.

Return value: text

The prototype of the DBE\_SCHEDULER.RUN\_FOREGROUND\_JOB function is as follows:

```

DBE_SCHEDULER.run_foreground_job(
job_name text
)return text

```

Example:

```

gaussdb=# CREATE USER test1 IDENTIFIED BY '*****';
NOTICE: The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
gaussdb=# SELECT DBE_SCHEDULER.create_credential('cre_1', 'test1', '*****');
create_credential

(1 row)

gaussdb=# SELECT DBE_SCHEDULER.create_job(job_name=>'job1', job_type=>'EXTERNAL_SCRIPT',
job_action=>'/usr/bin/pwd', enabled=>true, auto_drop=>false, credential_name => 'cre_1');
create_job

(1 row)

gaussdb=# CALL DBE_SCHEDULER.run_foreground_job('job1');
run_foreground_job

Host key verification failed.\r+
(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
drop_credential

(1 row)

gaussdb=# DROP USER test1;
DROP ROLE

```

- **DBE\_SCHEDULER.STOP\_JOB**

Stops a scheduled job.

The prototype of the DBE\_SCHEDULER.STOP\_JOB function is as follows:

```

DBE_SCHEDULER.stop_job(
job_name text,
force boolean default false,
commit_semantics text default 'STOP_ON_FIRST_ERROR'
)

```

**Table 10-325** DBE\_SCHEDULER.STOP\_JOB API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                          |
|-----------|------|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| job_name  | text | IN                     | No           | Name or class of a scheduled job. You can specify one or more scheduled jobs. If you specify multiple scheduled jobs, separate them with commas (,). |

| Parameter        | Type    | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                                                                                                                                                                                     |
|------------------|---------|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| force            | boolean | IN                     | No           | Specifies whether to delete a scheduled job. <ul style="list-style-type: none"> <li>• <b>true</b>: The scheduler sends a termination signal to end the job thread immediately.</li> <li>• <b>false</b>: The scheduler attempts to use the interrupt signal to terminate the scheduled job thread.</li> </ul>                    |
| commit_semantics | text    | IN                     | No           | Commit rules: <ul style="list-style-type: none"> <li>• <b>'STOP_ON_FIRST_ERROR'</b>: The interrupt operation performed before the first error is reported is committed.</li> <li>• <b>'ABSORB_ERRORS'</b>: The system attempts to bypass an error and commit the interrupt operation that is performed successfully.</li> </ul> |

Example:

```
gaussdb=# SELECT db_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
create_job
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.stop_job('job1', true, 'STOP_ON_FIRST_ERROR');
stop_job
```

-----

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
```

-----

(1 row)

- **DBE\_SCHEDULER.STOP\_SINGLE\_JOB**

Stops a single scheduled job.

The prototype of the DBE\_SCHEDULER.STOP\_SINGLE\_JOB function is as follows:

```
DBE_SCHEDULER.stop_single_job(
job_name text,
```

```
force boolean default false
)
```

**Example:**

```
gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;','0,null,null,null','DEFAULT_JOB_CLASS',false,false,null,null,null);
create_job

```

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.stop_single_job('job1', true);
stop_single_job

```

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job

```

(1 row)

- **DBE\_SCHEDULER.GENERATE\_JOB\_NAME**

Generates the name of a scheduled job.

The prototype of the DBE\_SCHEDULER.GENERATE\_JOB\_NAME function is as follows:

```
DBE_SCHEDULER.generate_job_name(
prefix text default 'JOB$_'
)return text
```

**Table 10-326** DBE\_SCHEDULER.GENERATE\_JOB\_NAME API parameters

| Parameter | Type | Input/Output Parameter | Can Be Empty | Description                                                                                                                                                     |
|-----------|------|------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| prefix    | text | IN                     | No           | Prefix of the generated name. The default value is 'JOB\$_'. Scheduled jobs that are repeatedly executed are named as follows:<br>job\$_1, job\$_2, job\$_3 ... |

**Example:**

```
gaussdb=# CALL DBE_SCHEDULER.generate_job_name();
generate_job_name

```

JOB\$\_1  
(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.generate_job_name();
generate_job_name

```

```
JOB$_2
(1 row)

gaussdb=# CALL DBE_SCHEDULER.generate_job_name('job');
generate_job_name

job3
(1 row)

gaussdb=# CALL DBE_SCHEDULER.generate_job_name('job');
generate_job_name

job4
(1 row)
```

### NOTICE

When DBE\_SCHEDULER.GENERATE\_JOB\_NAME is executed for the first time, a temporary sequence is created in **public** to store the sequence number of the current name. A common user does not have the create permission in **public**. Therefore, if a common user calls the function for the first time in the current database, the function fails to be called. In this case, you need to grant the create permission in **public** to the common user or call the API as a user with the create permission to create a temporary sequence.

- DBE\_SCHEDULER.CREATE\_PROGRAM

Creates a program.

The prototype of the DBE\_SCHEDULER.CREATE\_PROGRAM function is as follows:

```
DBE_SCHEDULER.create_program(
program_name text,
program_type text,
program_action text,
number_of_arguments integer default 0,
enabled boolean default false,
comments text default NULL
)
```

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 3, false, 'test');
create_program

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program

(1 row)
```

- DBE\_SCHEDULER.DEFINE\_PROGRAM\_ARGUMENT

Defines program parameters.

An API with the default value **default value** will not convert characters to lowercase letters by default. In this version, characters are case-sensitive.

The prototype of the DBE\_SCHEDULER.DEFINE\_PROGRAM\_ARGUMENT function is as follows:

```
DBE_SCHEDULER.define_program_argument(
program_name text,
```

```

argument_position integer,
argument_name text default NULL,
argument_type text,
out_argument boolean default false
)

-- With a default value --
DBE_SCHEDULER.define_program_argument(
program_name text,
argument_position integer,
argument_name text default NULL,
argument_type text,
default_value text,
out_argument boolean default false
)

```

**Example:**

```

gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 2, false, 'test');
create_program

(1 row)

gaussdb=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', false);
define_program_argument

(1 row)

gaussdb=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', 'value1',
false);
define_program_argument

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program

(1 row)

```

- **DBE\_SCHEDULER.DROP\_PROGRAM**

Deletes a program.

The prototype of the DBE\_SCHEDULER.DROP\_PROGRAM function is as follows:

```

DBE_SCHEDULER.drop_program(
program_name text,
force boolean default false
)

```

**Example:**

```

gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 2, false, 'test');
create_program

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program

(1 row)

```

- **DBE\_SCHEDULER.DROP\_SINGLE\_PROGRAM**

Deletes a single program.

The prototype of the DBE\_SCHEDULER.DROP\_SINGLE\_PROGRAM function is as follows:

```
DBE_SCHEDULER.drop_single_program(
program_name text,
force boolean default false
)
```

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 2, false, 'test');
create_program
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_single_program('program1', false);
drop_single_program
```

```

(1 row)
```

- DBE\_SCHEDULER.SET\_JOB\_ARGUMENT\_VALUE

Sets the parameters of a scheduled job. **argument\_value** can be left empty.

The prototype of the DBE\_SCHEDULER.SET\_JOB\_ARGUMENT\_VALUE function is as follows:

```
DBE_SCHEDULER.set_job_argument_value(
job_name text,
argument_position integer,
argument_value text
)
```

```
DBE_SCHEDULER.set_job_argument_value(
job_name text,
argument_name text,
argument_value text
)
```

Example:

```
gaussdb=# CALL dbe_scheduler.create_job('job1','EXTERNAL_SCRIPT','BEGIN INSERT INTO test1
VALUES(12); end;',2,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
create_job
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.set_job_argument_value('job1', 1, 'value1');
set_job_argument_value
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
```

```

(1 row)
```

- DBE\_SCHEDULER.CREATE\_SCHEDULE

Creates a schedule.

The prototype of the DBE\_SCHEDULER.CREATE\_SCHEDULE function is as follows:

```
DBE_SCHEDULER.create_schedule(
schedule_name text,
```

```
start_date timestamp with time zone default NULL,
repeat_interval text,
end_date timestamp with time zone default NULL,
comments text default NULL
)
```

**Example:**

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
null, 'test1');
create_schedule
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
create_schedule
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
create_schedule
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
drop_single_schedule
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
drop_single_schedule
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule3', true);
drop_single_schedule
```

```

(1 row)
```

- **DBE\_SCHEDULER.DROP\_SCHEDULE**

Deletes a schedule.

The prototype of the DBE\_SCHEDULER.DROP\_SCHEDULE function is as follows:

```
DBE_SCHEDULER.drop_schedule(
schedule_name text,
force boolean default false
)
```

**Example:**

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
null, 'test1');
create_schedule
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
create_schedule
```

```

(1 row)
```

```

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
create_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
drop_single_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
drop_single_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule3', true);
drop_single_schedule

(1 row)

```

- **DBE\_SCHEDULER.DROP\_SINGLE\_SCHEDULE**

Deletes a single schedule.

The prototype of the DBE\_SCHEDULER.DROP\_SINGLE\_SCHEDULE function is as follows:

```

DBE_SCHEDULER.drop_single_schedule(
schedule_name text,
force boolean default false
)

```

Example:

```

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
null, 'test1');
create_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
create_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
create_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule1');
drop_single_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule2', false);
drop_single_schedule

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule3', true);

```

```
drop_single_schedule
```

```

(1 row)
```

- **DBE\_SCHEDULER.CREATE\_JOB\_CLASS**

Creates the class of a scheduled job.

The prototype of the DBE\_SCHEDULER.CREATE\_JOB\_CLASS function is as follows:

```
DBE_SCHEDULER.create_job_class(
job_class_name text,
resource_consumer_group text default NULL,
service text default NULL,
logging_level integer default 0,
log_history integer default NULL,
comments text default NULL
)
```

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
=> '123');
create_job_class
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_job_class('jc1', false);
drop_job_class
```

```

(1 row)
```

- **DBE\_SCHEDULER.DROP\_JOB\_CLASS**

Deletes the class of a scheduled job.

The prototype of the DBE\_SCHEDULER.DROP\_JOB\_CLASS function is as follows:

```
DBE_SCHEDULER.drop_job_class(
job_class_name text,
force boolean default false
)
```

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
=> '123');
create_job_class
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_job_class('jc1', false);
drop_job_class
```

```

(1 row)
```

- **DBE\_SCHEDULER.DROP\_SINGLE\_JOB\_CLASS**

Deletes the class of a single scheduled job.

The prototype of the DBE\_SCHEDULER.DROP\_SINGLE\_JOB\_CLASS function is as follows:

```
DBE_SCHEDULER.drop_single_job_class(
job_class_name text,
force boolean default false
)
```

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
=> '123');
create_job_class

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_job_class('jc1', false);
drop_single_job_class

(1 row)
```

- **DBE\_SCHEDULER.GRANT\_USER\_AUTHORIZATION**

Grants the scheduled job permissions to the database user. The user who calls this function must have the SYSADMIN permission.

The prototype of the DBE\_SCHEDULER.GRANT\_USER\_AUTHORIZATION function is as follows:

```
DBE_SCHEDULER.grant_user_authorization(
username text,
privilege text
)
```

Example:

```
gaussdb=# CREATE USER user1 PASSWORD '1*s*****';
NOTICE: The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
gaussdb=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'create job');
grant_user_authorization

(1 row)
gaussdb=# DROP USER user1;
DROP ROLE
```

- **DBE\_SCHEDULER.REVOKE\_USER\_AUTHORIZATION**

Revokes the scheduled job permissions from the database user. The user who calls this function must have the SYSADMIN permission.

The prototype of the DBE\_SCHEDULER.REVOKE\_USER\_AUTHORIZATION function is as follows:

```
DBE_SCHEDULER.revoke_user_authorization(
username text,
privilege text
)
```

Example:

```
gaussdb=# CREATE USER user1 PASSWORD '1*s*****';
NOTICE: The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
gaussdb=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'CREATE JOB');
grant_user_authorization

(1 row)

gaussdb=# CALL DBE_SCHEDULER.revoke_user_authorization('user1', 'CREATE JOB');
revoke_user_authorization

(1 row)
gaussdb=# DROP USER user1;
DROP ROLE
```

- **DBE\_SCHEDULER.CREATE\_CREDENTIAL**

Creates an authorization certificate. The user who calls this function must have the SYSADMIN permission.

The prototype of the DBE\_SCHEDULER.CREATE\_CREDENTIAL function is as follows:

```
DBE_SCHEDULER.create_credential(
credential_name text,
username text,
password text default NULL,
database_role text default NULL,
windows_domain text default NULL,
comments text default NULL
)
```

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
create_credential

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
drop_credential

(1 row)
```

#### NOTICE

The **password** parameter of DBE\_SCHEDULER.CREATE\_CREDENTIAL must be set to **NULL** or '\*\*\*\*\*'. This parameter is used only for compatibility and does not indicate any actual meaning. Do not use the OS username corresponding to the installation user to create a certificate.

- DBE\_SCHEDULER.DROP\_CREDENTIAL

Destroys an authorization certificate. The user who calls this function must have the SYSADMIN permission.

The prototype of the DBE\_SCHEDULER.DROP\_CREDENTIAL function is as follows:

```
DBE_SCHEDULER.drop_credential(
credential_name text,
force boolean default false
)
```

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
create_credential

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
drop_credential

(1 row)
```

- DBE\_SCHEDULER.ENABLE

Enables an object.

The prototype of the DBE\_SCHEDULER.ENABLE function is as follows:

```
DBE_SCHEDULER.enable(
name text,
commit_semantics text default 'STOP_ON_FIRST_ERROR'
)
```

**Example:**

```
gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;','0,null,null,null','DEFAULT_JOB_CLASS',false,false,null,null,null);
create_job
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'INSERT INTO
tb_job_test(key) VALUES(null);', 0, false, '');
create_program
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.enable('job1');
enable
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.enable('program1', 'STOP_ON_FIRST_ERROR');
enable
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program
```

```

(1 row)
```

- **DBE\_SCHEDULER.ENABLE\_SINGLE**

Enables a single object.

The prototype of the DBE\_SCHEDULER.ENABLE\_SINGLE function is as follows:

```
DBE_SCHEDULER.enable_single(
name text
)
```

**Example:**

```
gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;','0,null,null,null','DEFAULT_JOB_CLASS',false,false,null,null,null);
create_job
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.enable_single('job1');
enable_single
```

```

(1 row)
```

```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
```

```

```

(1 row)

- **DBE\_SCHEDULER.DISABLE**

Disables multiple objects. The value of name is a character string separated by commas (,). Each character string separated by commas (,) is an object.

The prototype of the DBE\_SCHEDULER.DISABLE function is as follows:

```
DBE_SCHEDULER.disable(
name text,
force boolean default false,
commit_semantics text default 'STOP_ON_FIRST_ERROR'
)
```

Example:

```
gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;','0,null,null,null','DEFAULT_JOB_CLASS',false,false,null,null,null);
create_job

```

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'INSERT INTO
tb_job_test(key) VALUES(null);', 0, false, '');
create_program

```

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.disable('job1');
disable

```

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.disable('program1', false, 'STOP_ON_FIRST_ERROR');
disable

```

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job

```

(1 row)

```
gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
drop_program

```

(1 row)

- **DBE\_SCHEDULER.DISABLE\_SINGLE**

Disables a single object.

The prototype of the DBE\_SCHEDULER.DISABLE\_SINGLE function is as follows:

```
DBE_SCHEDULER.disable_single(
name text,
force boolean default false
)
```

Example:

```
gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;','0,null,null,null','DEFAULT_JOB_CLASS',false,false,null,null,null);
create_job

```

```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.disable_single('job1', false);
disable_single

```

```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job

```

```
(1 row)
```

- **DBE\_SCHEDULER.EVAL\_CALENDAR\_STRING**

Analyzes the scheduling job period.

Return type: timestamp with time zone

The prototype of the DBE\_SCHEDULER.EVAL\_CALENDAR\_STRING function is as follows:

```
DBE_SCHEDULER.evaluate_calendar_string(
IN calendar_string text,
IN start_date timestamp with time zone,
IN return_date_after timestamp with time zone
)return timestamp with time zone
```

Example:

```
gaussdb=# CALL DBE_SCHEDULER.eval_calendar_string('FREQ=DAILY; BYHOUR=6;', sysdate, sysdate);
eval_calendar_string

2023-09-15 06:47:24+08
(1 row)
```

- **DBE\_SCHEDULER.EVALUATE\_CALENDAR\_STRING**

Analyzes the scheduling job period.

The prototype of the DBE\_SCHEDULER.EVALUATE\_CALENDAR\_STRING function is as follows:

```
DBE_SCHEDULER.evaluate_calendar_string(
IN calendar_string text,
IN start_date timestamp with time zone,
IN return_date_after timestamp with time zone,
OUT next_run_date timestamp with time zone
)return timestamp with time zone
```

Example:

```
gaussdb=# CREATE OR REPLACE PROCEDURE pr1(calendar_str text) as
DECLARE
start_date timestamp with time zone;
return_date_after timestamp with time zone;
next_run_date timestamp with time zone;
BEGIN
start_date := '2003-2-1 10:30:00.111111+8'::timestamp with time zone;
return_date_after := start_date;
DBE_SCHEDULER.evaluate_calendar_string(
calendar_str,
start_date, return_date_after, next_run_date);
DBE_OUTPUT.PRINT_LINE('next_run_date: ' || next_run_date);
return_date_after := next_run_date;
END;
/
CREATE PROCEDURE

gaussdb=# CALL pr1('FREQ=hourly;INTERVAL=2;BYHOUR=6,10;BYMINUTE=0;BYSECOND=0');
next_run_date: 2003-02-02 06:00:00+08
pr1

```

(1 row)

## 10.11.2.12 DBE\_APPLICATION\_INFO

### APIs

**Table 10-327** provides all APIs supported by the **DBE\_APPLICATION\_INFO** package. **DBE\_APPLICATION\_INFO** applies to the current session.

**Table 10-327** DBE\_APPLICATION\_INFO

| API                                                   | Description                |
|-------------------------------------------------------|----------------------------|
| <a href="#">DBE_APPLICATION_INFO.SET_CLIENT_INFO</a>  | Writes client information. |
| <a href="#">DBE_APPLICATION_INFO.READ_CLIENT_INFO</a> | Reads client information.  |

- [DBE\\_APPLICATION\\_INFO.SET\\_CLIENT\\_INFO](#)

Writes client information. The [DBE\\_APPLICATION\\_INFO.SET\\_CLIENT\\_INFO](#) function prototype is as follows:

```
DBE_APPLICATION_INFO.SET_CLIENT_INFO(
 str text
)returns void;
```

**Table 10-328** DBE\_APPLICATION\_INFO.SET\_CLIENT\_INFO API parameters

| Parameter | Description                |
|-----------|----------------------------|
| str       | Writes client information. |

- [DBE\\_APPLICATION\\_INFO.READ\\_CLIENT\\_INFO](#)

The [DBE\\_APPLICATION\\_INFO.READ\\_CLIENT\\_INFO](#) function prototype is as follows:

```
DBE_APPLICATION_INFO.READ_CLIENT_INFO(
 OUT client_info text);
```

**Table 10-329** DBE\_APPLICATION\_INFO.READ\_CLIENT\_INFO API parameters

| Parameter   | Description        |
|-------------|--------------------|
| client_info | Client information |

### 10.11.2.13 DBE\_XMLDOM

#### APIs

The advanced function package **DBE\_XMLDOM** is used to access XMLType objects and implement Document Object Model (DOM), which is an API used to access HTML and XML documents. For details about all types supported by the advanced function package **DBE\_XMLDOM**, see [Table 10-330](#). For details about all APIs supported by **DBE\_XMLDOM**, see [Table 10-331](#).

#### NOTE

When the **DBE\_XMLDOM** advanced package is used in the database whose character set is set to **SQL\_ASCII**, an error is reported if characters beyond the ASCII range are input.

**Table 10-330** DBE\_XMLDOM data types

| Type            | Description                         |
|-----------------|-------------------------------------|
| DOMATTR         | Implements the DOMAttributes API.   |
| DOMDOCUMENT     | Implements the DOMDocument API.     |
| DOMELEMENT      | Implements the DOMELEMENT API.      |
| DOMNAMEDNODEMAP | Implements the DOMNamedNodeMap API. |
| DOMNODELIST     | Implements the DOMNodeList API.     |
| DOMNODE         | Implements the DOMNode API.         |
| DOMTEXT         | Implements the DOMText API.         |

**Table 10-331** DBE\_XMLDOM parameters

| API                                       | Description                                                                               |
|-------------------------------------------|-------------------------------------------------------------------------------------------|
| <a href="#">DBE_XMLDOM.APPENDCHILD</a>    | Adds the newchild node to the end of the parent(n) node and returns the newly added node. |
| <a href="#">DBE_XMLDOM.CREATEELEMENT</a>  | Creates a DOMELEMENT object with the specified name.                                      |
| <a href="#">DBE_XMLDOM.CREATETEXTNODE</a> | Creates a DOMText node.                                                                   |
| <a href="#">DBE_XMLDOM.FREEDOCUMENT</a>   | Frees resources related to DOMDocument nodes.                                             |
| <a href="#">DBE_XMLDOM.FREEELEMENT</a>    | Frees resources related to DOMELEMENT nodes.                                              |
| <a href="#">DBE_XMLDOM.FREENODE</a>       | Frees resources related to DOMNode nodes.                                                 |

| API                                             | Description                                                         |
|-------------------------------------------------|---------------------------------------------------------------------|
| <a href="#">DBE_XMLDOM.FREENODELIST</a>         | Frees resources related to DOMNodeList nodes.                       |
| <a href="#">DBE_XMLDOM.GETATTRIBUTE</a>         | Returns the attribute values of a DOMELEMENT object by name.        |
| <a href="#">DBE_XMLDOM.GETATTRIBUTES</a>        | Returns the attribute values of a DOMNode node as a map.            |
| <a href="#">DBE_XMLDOM.GETCHILDNODES</a>        | Converts several subnodes under a node into a node list.            |
| <a href="#">DBE_XMLDOM.GETCHILDRENBYTAGNAME</a> | Returns the subnodes of a DOMELEMENT node by name.                  |
| <a href="#">DBE_XMLDOM.GETDOCUMENTELEMENT</a>   | Returns the first subnode of the specified document.                |
| <a href="#">DBE_XMLDOM.GETFIRSTCHILD</a>        | Returns the first subnode.                                          |
| <a href="#">DBE_XMLDOM.GETLASTCHILD</a>         | Returns the last subnode.                                           |
| <a href="#">DBE_XMLDOM.GETLENGTH</a>            | Obtains the number of subnodes under a specified node.              |
| <a href="#">DBE_XMLDOM.GETLOCALNAME</a>         | Returns the local name of a node.                                   |
| <a href="#">DBE_XMLDOM.GETNAMEDITEM</a>         | Returns the node specified by name.                                 |
| <a href="#">DBE_XMLDOM.GETNEXTSIBLING</a>       | Returns the next node of the specified node.                        |
| <a href="#">DBE_XMLDOM.GETNODENAME</a>          | Returns the name of a node.                                         |
| <a href="#">DBE_XMLDOM.GETNODETYPE</a>          | Returns the type of a node.                                         |
| <a href="#">DBE_XMLDOM.GETNODEVALUE</a>         | Obtains the value of a node, depending on its type.                 |
| <a href="#">DBE_XMLDOM.GETPARENTNODE</a>        | Returns the parent node of a node.                                  |
| <a href="#">DBE_XMLDOM.GETTAGNAME</a>           | Returns the tag name of the specified DOMELEMENT node.              |
| <a href="#">DBE_XMLDOM.HASCHILDNODES</a>        | Checks whether the DOMNode object has any subnode.                  |
| <a href="#">DBE_XMLDOM.IMPORTNODE</a>           | Copies a node and specifies the document to which the node belongs. |

| API                                      | Description                                                                                            |
|------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <b>DBE_XMLDOM.ISNULL</b>                 | Checks whether a node is null.                                                                         |
| <b>DBE_XMLDOM.ITEM</b>                   | Returns the item corresponding to the index parameter in the mapping.                                  |
| <b>DBE_XMLDOM.MAKEELEMENT</b>            | Converts a DOMNode object to the DOMELEMENT type.                                                      |
| <b>DBE_XMLDOM.MAKENODE</b>               | Forcibly converts a node to the DOMNode type.                                                          |
| <b>DBE_XMLDOM.NEWDOMDOCUMENT</b>         | Returns a new DOMDocument object.                                                                      |
| <b>DBE_XMLDOM.SETATTRIBUTE</b>           | Sets the value of the DOMELEMENT attribute by name.                                                    |
| <b>DBE_XMLDOM.SETCHARSET</b>             | Sets the character set for a DOMDocument object.                                                       |
| <b>DBE_XMLDOM.SETDOCTYPE</b>             | Sets the external DTD of a DOMDocument object.                                                         |
| <b>DBE_XMLDOM.SETNODEVALUE</b>           | Sets the value of a node in the DOMNode object.                                                        |
| <b>DBE_XMLDOM.WRITETOBUFFER</b>          | Writes an XML node to a specified buffer.                                                              |
| <b>DBE_XMLDOM.WRITETOCLOB</b>            | Writes an XML node to a specified CLOB.                                                                |
| <b>DBE_XMLDOM.WRITETOFILE</b>            | Writes an XML node to a specified file.                                                                |
| <b>DBE_XMLDOM.GETSESSIONTREECOUNT</b>    | Displays the number of DOM trees of all types in the current session.                                  |
| <b>DBE_XMLDOM.GETDOCTREEINFO</b>         | Displays statistics such as the memory usage and number of nodes of the DOM tree of the document type. |
| <b>DBE_XMLDOM.GETDETAILEDDOCTREEINFO</b> | Displays the number of nodes of each type for a specific document variable.                            |

- **DBE\_XMLDOM.APPENDCHILD**

Adds the newchild node to the end of the parent(n) node and returns the newly added node. The prototype of the DBE\_XMLDOM.APPENDCHILD function is as follows:

```
DBE_XMLDOM.APPENDCHILD(
 n IN DOMNode,
 newchild IN DOMNode)
RETURN DOMNODE;
```

**Table 10-332** DBE\_XMLDOM.APPENDCHILD parameters

| Parameter | Description      |
|-----------|------------------|
| n         | Node to be added |
| newchild  | New node added   |

 **NOTE**

1. The error message "operation not support" is displayed for the APPEND ATTR node under the DOCUMENT node. Database ORA does not report an error in this scenario, but the mounting fails.
2. The error message "operation not support" is displayed for the APPEND ATTR node under the ATTR node. Database ORA does not report an error in this scenario, but the mounting fails.
3. When multiple child nodes of the ATTR type are added to a parent node, the child nodes with the same key value cannot exist under the same parent node.

**Example:**

-- Add a DOMNode node to a specified DOC tree and use DBE\_XMLDOM.HASCHILDNODES() to check whether the subnode is successfully added.

```

DECLARE
 doc DBE_XMLDOM.DOMDocument;
 doc1 DBE_XMLDOM.DOMDocument;
 root DBE_XMLDOM.DOMELEMENT;
 rootnode DBE_XMLDOM.DOMNode;
 child1 DBE_XMLDOM.DOMELEMENT;
 child2 DBE_XMLDOM.DOMELEMENT;
 attr DBE_XMLDOM.DOMAttr;
 text DBE_XMLDOM.DOMTEXT;
 node DBE_XMLDOM.DOMNode;
 child1_node DBE_XMLDOM.DOMNode;
 attr_node DBE_XMLDOM.DOMNode;
 parent DBE_XMLDOM.DOMNode;
 buf varchar2(1000);
BEGIN
 doc := DBE_XMLDOM.newDOMDocument();
 root := DBE_XMLDOM.createElement(doc, 'root');
 rootnode := DBE_xmlldom.makeNode(root);
 node := DBE_XMLDOM.appendChild(DBE_xmlldom.makeNode(doc), rootnode);
 child1 := DBE_XMLDOM.createElement(doc, 'child1');
 child1_node := DBE_XMLDOM.makeNode(child1);
 node := DBE_XMLDOM.appendChild(rootnode, child1_node);
 attr := DBE_XMLDOM.createAttribute(doc, 'abc');
 attr_node := DBE_XMLDOM.makeNode(attr);
 node := DBE_XMLDOM.appendChild(child1_node, attr_node);
 IF DBE_XMLDOM.HASCHILDNODES(child1_node) THEN
 DBE_OUTPUT.print_line('HAS CHILD NODES');
 ELSE
 DBE_OUTPUT.print_line('NOT HAS CHILD NODES ');
 END IF;
 parent := DBE_XMLDOM.GETPARENTNODE(attr_node);
 buf := DBE_XMLDOM.GETNODENAME(parent);
 DBE_OUTPUT.print_line(buf);
END;
/

```

- **DBE\_XMLDOM.CREATEELEMENT**

Returns the DOMELEMENT object with the specified name. The prototype of the DBE\_XMLDOM.CREATEELEMENT function is as follows:

```

DBE_XMLDOM.CREATEELEMENT(
 doc IN DOMDOCUMENT,

```

```
tagName IN VARCHAR2)
RETURN DOMELEMENT;
```

Returns the DOMELEMENT object with the specified name and namespace. The prototype of the DBE\_XMLDOM.CREATEELEMENT function is as follows:

```
DBE_XMLDOM.CREATEELEMENT(
doc IN DOMDOCUMENT,
tagName IN VARCHAR2,
ns IN VARCHAR2)
RETURN DOMELEMENT;
```

**Table 10-333** DBE\_XMLDOM.CREATEELEMENT parameters

| Parameter | Description                       |
|-----------|-----------------------------------|
| doc       | Specified DOMDocument object      |
| tagName   | Name of the new DOMELEMENT object |
| ns        | Namespace                         |

 **NOTE**

1. When the **tagName** parameter is set to null or an empty character string, the exception "NULL or invalid TagName argument specified" is thrown.
2. The default maximum length of **tagName** and **ns** is 32767. If the length exceeds 32767, an exception is thrown.

**Example:**

--1. Create a DOMELEMENT object with the specified name.

```
DECLARE
doc dbe_xmldom.domdocument;
attr DBE_XMLDOM.DOMATTR;
elem DBE_XMLDOM.DOMELEMENT;
ans DBE_XMLDOM.DOMATTR;
buf varchar2(1010);
BEGIN
doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
<computer size="ITX"><cpu>Ryzen 9 3950X</cpu>
<ram>32GBx2 DDR4 3200MHz</ram>
<motherboard>ROG X570i</motherboard>
<gpu>RTX2070 Super</gpu>
<ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
<hdd>12TB WD Digital</hdd>
<psu>CORSAIR SF750</psu>
<case>LIANLI TU150</case>
</computer>');
elem := dbe_xmldom.createelement(doc,'elem');
DBE_XMLDOM.WRITETOBUFFER(dbe_xmldom.makenode(elem), buf);
DBE_OUTPUT.print_line(buf);
END;
/
```

--2. Create a DOMELEMENT object with the specified name and namespace.

```
DECLARE
doc dbe_xmldom.domdocument;
attr DBE_XMLDOM.DOMATTR;
elem DBE_XMLDOM.DOMELEMENT;
ans DBE_XMLDOM.DOMNODE;
buf varchar2(1010);
list DBE_XMLDOM.DOMNODELIST;
node DBE_XMLDOM.DOMNODE;
BEGIN
```

```

doc := dbe_xmlDOM.newDOMdocument('<h:data xmlns:h="http://www.w3.org/TR/html4/">
 <h:da1 len="10">test namespace</h:da1><h:da1>bbbbbbbbbb</h:da1></h:data>');
elem := dbe_xmlDOM.createelement(doc,'elem','http://www.w3.org/TR/html5/');
ans := DBE_XMLDOM.APPENDCHILD(dbe_xmlDOM.makenode(doc), dbe_xmlDOM.makenode(elem));
DBE_XMLDOM.WRITETOBUFFER(doc, buf);
DBE_OUTPUT.print_line(buf);
END;
/

```

- **DBE\_XMLDOM.CREATETEXTNODE**

Creates and returns a DOMText object. The prototype of the DBE\_XMLDOM.CREATETEXTNOD function is as follows:

```

DBE_XMLDOM.CREATETEXTNODE(
 doc IN DOMDocument,
 data IN VARCHAR2)
RETURN DOMTEXT;

```

**Table 10-334** DBE\_XMLDOM.CREATETEXTNODE parameters

| Parameter | Description                  |
|-----------|------------------------------|
| doc       | Specified DOMDocument object |
| data      | Content of the DOMText node  |

 **NOTE**

1. You can enter an empty string or null value for **data**.
  2. The default maximum length of **data** is 32767. If the length exceeds 32767, an exception is thrown.
- **DBE\_XMLDOM.FREEDOCUMENT**

Frees a DOMDocument node. The prototype of the DBE\_XMLDOM.FREEDOCUMENT function is as follows:

```

DBE_XMLDOM.FREEDOCUMENT(
 doc IN DOMDOCUMENT);

```

**Table 10-335** DBE\_XMLDOM.FREEDOCUMENT parameters

| Parameter | Description                |
|-----------|----------------------------|
| doc       | Specified DOMDocument node |

**Example:**

```

-- Free the entire DOC tree after the DOMNode node is added to the DOC tree.
DECLARE
 doc dbe_xmlDOM.domdocument;
 elem dbe_xmlDOM.domelement;
 doc_node dbe_xmlDOM.DOMNODE;
 root_elmt dbe_xmlDOM.DOMELEMENT;
 root_node dbe_xmlDOM.DOMNODE;
 value varchar(1000);
BEGIN
 doc := dbe_xmlDOM.newDOMdocument();
 doc_node := dbe_xmlDOM.MAKENODE(doc);
 root_elmt := dbe_xmlDOM.CREATEELEMENT(doc,'staff');
 root_node:=dbe_xmlDOM.APPENDCHILD(doc_node, dbe_xmlDOM.MAKENODE(root_elmt));

```

```
dbe_xmlDOM.freedocument(doc);
END;
/
```

- DBE\_XMLDOM.FREEELEMENT

Frees a DOMELEMENT node. The prototype of the DBE\_XMLDOM.FREEELEMENT function is as follows:

```
DBE_XMLDOM.FREEELEMENT(
elem IN DOMELEMENT);
```

**Table 10-336** DBE\_XMLDOM.FREEELEMENT parameters

| Parameter | Description               |
|-----------|---------------------------|
| elem      | Specified DOMELEMENT node |

- DBE\_XMLDOM.FREENODE

Frees a DOMNode node. The prototype of the DBE\_XMLDOM.FREENODE function is as follows:

```
DBE_XMLDOM.FREENODE(
n IN DOMNODE);
```

**Table 10-337** DBE\_XMLDOM.FREENODE parameters

| Parameter | Description            |
|-----------|------------------------|
| n         | Specified DOMNode node |

 **NOTE**

1. After GaussDB performs the FREENODE operation, the freed node is not available again. After the database ORA performs the FREENODE operation, the freed node is available again and becomes another node.
  2. When other APIs call the freed DOMNode node, the calling is different from that in database ORA.
- DBE\_XMLDOM.FREENODELIST

Frees a DOMNodeList node. The prototype of the DBE\_XMLDOM.FREENODE function is as follows:

```
DBE_XMLDOM.GETLENGTH(
nl IN DOMNODELIST);
```

**Table 10-338** DBE\_XMLDOM.FREENODELIST parameters

| Parameter | Description                |
|-----------|----------------------------|
| nl        | Specified DOMNodeList node |

 **NOTE**

1. A DOMNodeList node will be completed freed by FREENODELIST.
  2. When other APIs call the freed DOMNodeList node, the calling is different from that in database ORA.
  3. The input parameter **freenodelist** cannot be empty.
- **DBE\_XMLDOM.GETATTRIBUTE**

Returns the attribute values of a DOMELEMENT object by name. The prototype of the DBE\_XMLDOM.GETATTRIBUTE function is as follows:

```
DBE_XMLDOM.GETATTRIBUTE(
 elem IN DOMELEMENT,
 name IN VARCHAR2)
RETURN VARCHAR2;
```

Returns the attribute values of a DOMELEMENT object by name and namespace URI. The prototype of the DBE\_XMLDOM.GETATTRIBUTE function is as follows:

```
DBE_XMLDOM.GETATTRIBUTE(
 elem IN DOMELEMENT,
 name IN VARCHAR2,
 ns IN VARCHAR2)
RETURN VARCHAR2;
```

**Table 10-339** DBE\_XMLDOM.GETATTRIBUTE parameters

| Parameter | Description               |
|-----------|---------------------------|
| elem      | Specified DOMELEMENT node |
| name      | Attribute name            |
| ns        | Namespace                 |

 **NOTE**

1. The **ns** parameter of the DBE\_XMLDOM.GETATTRIBUTE API does not support the asterisk (\*) parameter.
2. GaussDB does not support the namespace prefix as an attribute, and the value of the prefix cannot be queried through the DBE\_XMLDOM.GETATTRIBUTE API.

**Example:**

```
--1. Return the attribute values of a DOMELEMENT object by name.
DECLARE
 doc dbe_xmldom.domdocument;
 elem dbe_xmldom.domelement;
 docnode DBE_XMLDOM.DOMNode;
 buffer varchar2(1010);
 value varchar2(1000);
BEGIN
 doc := dbe_xmldom.newDOMDocument();
 elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
 DBE_XMLDOM.setattribute(elem, 'len', '50cm');
 docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
 DBE_XMLDOM.makeNode(elem));
 value := DBE_XMLDOM.getattribute(elem, 'len');
 dbe_output.print_line('value: ');
 dbe_output.print_line(value);
 dbe_xmldom.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
```

```

END;
/

--2. Return the attribute values of a DOMELEMENT object by name and namespace URI.
DECLARE
 doc dbe_xmlDOM.domdocument;
 elem dbe_xmlDOM.domelement;
 docnode DBE_XMLDOM.DOMNode;
 buffer varchar2(1010);
 value varchar(1000);

BEGIN
 doc := dbe_xmlDOM.newDOMDocument();
 elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
 DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
 docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
 DBE_XMLDOM.makeNode(elem));
 value := DBE_XMLDOM.getattribute(elem, 'len', 'www.huawei.com');
 dbe_output.print_line('value: ');
 dbe_output.print_line(value);
 dbe_xmlDOM.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
END;
/

```

- **DBE\_XMLDOM.GETATTRIBUTES**

Returns the attribute values of a DOMNode node as a map. The prototype of the DBE\_XMLDOM.GETATTRIBUTES function is as follows:

```

DBE_XMLDOM.GETATTRIBUTES(
 n IN DOMNode)
RETURN DOMNAMEDNODEMAP;

```

**Table 10-340** DBE\_XMLDOM.GETATTRIBUTES parameters

| Parameter | Description            |
|-----------|------------------------|
| n         | Specified DOMNode node |

- **DBE\_XMLDOM.GETCHILDNODES**

Converts several subnodes under a node into a node list. The prototype of the DBE\_XMLDOM.GETCHILDNODES function is as follows:

```

DBE_XMLDOM.GETCHILDNODES(
 n IN DOMNode)
RETURN DOMNodeList;

```

**Table 10-341** DBE\_XMLDOM.GETCHILDNODES parameters

| Parameter | Description            |
|-----------|------------------------|
| n         | Specified DOMNode node |

- **DBE\_XMLDOM.GETCHILDRENBYTAGNAME**

Returns the subnodes of a DOMELEMENT node by name. The prototype of the DBE\_XMLDOM.GETCHILDRENBYTAGNAME function is as follows:

```

DBE_XMLDOM.GETCHILDRENBYTAGNAME (
 elem IN DOMELEMENT,
 name IN VARCHAR2)
RETURN DOMNODELIST;

```

Returns the subnodes of a DOMELEMENT node by name and namespace. The prototype of the DBE\_XMLDOM.GETCHILDRENBYTAGNAME function is as follows:

```
DBE_XMLDOM.GETCHILDRENBYTAGNAME (
 elem IN DOMELEMENT,
 name IN VARCHAR2,
 ns IN VARCHAR2)
RETURN DOMNODELIST;
```

**Table 10-342** DBE\_XMLDOM.GETCHILDRENBYTAGNAME parameters

| Parameter | Description               |
|-----------|---------------------------|
| elem      | Specified DOMELEMENT node |
| name      | Attribute name            |
| ns        | Namespace                 |

 **NOTE**

The **ns** parameter of the DBE\_XMLDOM.GETCHILDRENBYTAGNAME API does not support the asterisk (\*) parameter. To obtain all attributes of a node, use the DBE\_XMLDOM.GETCHILDNODES API.

**Example:**

```
--1. Return the subnodes of a DOMELEMENT node by name.
DECLARE
 doc dbe_xmldom.domdocument;
 elem dbe_xmldom.domelement;
 docnodelist dbe_xmldom.domnodelist;
 node_elem dbe_xmldom.domelement;
 node dbe_xmldom.domnode;
 buffer varchar2(1010);
 value varchar2(1000);
BEGIN
 doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
 <students age="16" hight="176">
 <student>
 <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
 </student>
 <student>
 <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
 </student>
 </students>');
 elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
 docnodelist := dbe_xmldom.GETCHILDRENBYTAGNAME(elem, 'student');
 node := dbe_xmldom.ITEM(docnodelist, 0);
 node_elem := dbe_xmldom.makeelement(node);
 value := DBE_XMLDOM.gettagname(node_elem);
 dbe_output.print_line('value: ');
 dbe_output.print_line(value);
 dbe_xmldom.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
END;
```

- **DBE\_XMLDOM.GETDOCUMENTELEMENT**

Returns the first subnode of the specified document. The prototype of the DBE\_XMLDOM.GETDOCUMENTELEMENT function is as follows:

```
DBE_XMLDOM.GETDOCUMENTELEMENT(
 doc IN DOMDOCUMENT)
RETURN DOMELEMENT;
```

**Table 10-343** DBE\_XMLDOM.GETDOCUMENTELEMENT parameters

| Parameter | Description                |
|-----------|----------------------------|
| doc       | Specified DOMDocument node |

**Example:**

```
-- Obtain the first subnode in the DOC tree and output the node name.
DECLARE
 doc dbe_xmldom.domdocument;
 elem dbe_xmldom.domelement;
 doc_node dbe_xmldom.DOMNODE;
 root_elmt dbe_xmldom.DOMELEMENT;
 root_node dbe_xmldom.DOMNODE;
 value varchar(1000);
BEGIN
 doc := dbe_xmldom.newdomdocument();
 doc_node := dbe_xmldom.MAKENODE(doc);
 root_elmt := dbe_xmldom.CREATEELEMENT(doc,'staff');
 root_node:=dbe_xmldom.APPENDCHILD(doc_node, dbe_xmldom.MAKENODE(root_elmt));
 elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
 value := DBE_XMLDOM.gettagname(elem);
 dbe_output.print_line(value);
END;
/
```

- **DBE\_XMLDOM.GETFIRSTCHILD**

Returns the first subnode of a node. The prototype of the DBE\_XMLDOM.GETFIRSTCHILD function is as follows:

```
DBE_XMLDOM.GETFIRSTCHILD(
 n IN DOMNODE)
RETURN DOMNODE;
```

**Table 10-344** DBE\_XMLDOM.GETFIRSTCHILD parameters

| Parameter | Description            |
|-----------|------------------------|
| n         | Specified DOMNode node |

**Example:**

```
-- Obtain the name and type of the first subnode after the DOC tree is converted to the DOMNode
type, and then obtain the name of the first subnode of the obtained first DOMNode node.
DECLARE
 doc dbe_xmldom.domdocument;
 doc_node dbe_xmldom.domnode;
 root_node dbe_xmldom.domnode;
 inside_node dbe_xmldom.domnode;
 node_name varchar2(1000);
 node_type integer;
BEGIN
 doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
 <students age="16" hight="176">
 <student1>
 <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
 </student1>
 <student2>
```

```

 <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
 </student2>
</students>');
doc_node := DBE_XMLDOM.MAKENODE(doc);
root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
node_name := DBE_XMLDOM.GETNODENAME(root_node);
node_type := DBE_XMLDOM.GETNODETYPE(root_node);
dbe_output.print_line(node_name);
dbe_output.print_line(node_type);
inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);
node_name := DBE_XMLDOM.GETNODENAME(inside_node);
dbe_output.print_line(node_name);
END;
/

```

- **DBE\_XMLDOM.GETLASTCHILD**

Returns the last subnode of a node. The prototype of the DBE\_XMLDOM.GETLASTCHILD function is as follows:

```

DBE_XMLDOM.GETLASTCHILD(
 n IN DOMNODE)
RETURN DOMNODE;

```

**Table 10-345** DBE\_XMLDOM.GETLASTCHILD parameters

| Parameter | Description            |
|-----------|------------------------|
| n         | Specified DOMNode node |

**Example:**

-- Obtain the name and type of the last subnode after the DOC tree is converted to the DOMNode type, and then obtain the name of the last subnode of the obtained last DOMNode node.

```

DECLARE
 doc dbe_xmldom.domdocument;
 doc_node dbe_xmldom.domnode;
 root_node dbe_xmldom.domnode;
 inside_node dbe_xmldom.domnode;
 node_name varchar2(1000);
 node_type integer;
BEGIN
 doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
 <students age="16" hight="176">
 <student1>
 <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
 </student1>
 <student2>
 <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
 </student2>
 </students>');
 doc_node := DBE_XMLDOM.MAKENODE(doc);
 root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
 node_name := DBE_XMLDOM.GETNODENAME(root_node);
 node_type := DBE_XMLDOM.GETNODETYPE(root_node);
 dbe_output.print_line(node_name);
 dbe_output.print_line(node_type);
 inside_node := DBE_XMLDOM.GETLASTCHILD(root_node);
 node_name := DBE_XMLDOM.GETNODENAME(inside_node);
 dbe_output.print_line(node_name);
END;
/

```

- **DBE\_XMLDOM.GETLENGTH**

Returns the number of subnodes under a DOMNamedNodeMap node. The prototype of the DBE\_XMLDOM.GETLENGTH function is as follows:

```
DBE_XMLDOM.GETLENGTH(
 nnm IN DOMNAMEDNODEMAP)
RETURN NUMBER;
```

Returns the number of subnodes under a DOMNodeList node. The prototype of the DBE\_XMLDOM.GETLENGTH function is as follows:

```
DBE_XMLDOM.GETLENGTH(
 nl IN DOMNODELIST)
RETURN NUMBER;
```

**Table 10-346** DBE\_XMLDOM.GETLENGTH parameters

| Parameter | Description                    |
|-----------|--------------------------------|
| nnm       | Specified DOMNamedNodeMap node |
| nl        | Specified DOMNodeList node     |

**Example:**

--1. Declare a DOMNamedNodeMap parameter in a function.

```
DECLARE
 doc DBE_XMLDOM.DOMDocument;
 elem DBE_XMLDOM.DOMELEMENT;
 map DBE_XMLDOM.DOMNAMEDNODEMAP;
 node DBE_XMLDOM.DOMNODE;
 buf varchar2(10000);
 len INTEGER;
BEGIN
 doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
 <bookstore category="web" cover="paperback">
 <book category="cooking">
 <title lang="en">Everyday Italian</title>
 <author>Giada De Laurentiis</author>
 <year>2005</year>
 <price>30.00</price>
 </book>
 </bookstore>');
 elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
 node := DBE_XMLDOM.MAKENODE(elem);
 map := DBE_XMLDOM.GETATTRIBUTES(node);
 len := DBE_XMLDOM.GETLENGTH(map);
 DBE_OUTPUT.print_line(len);
END;
/
```

--2. Declare a NodeList parameter in a function.

```
DECLARE
 doc dbe_xmldom.domdocument;
 node dbe_xmldom.domnode;
 node1 dbe_xmldom.domnode;
 nodelist DBE_XMLDOM.DOMNODELIST;
 len INTEGER;
 buffer1 varchar2(1010);
BEGIN
 doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
 <students age="16" hight="176">
 <student>
 <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
 </student>
 <student>
 <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
 </student>
 </students>');
 node := dbe_xmldom.makenode(doc);
```

```
node := dbe_xmlDOM.GETFIRSTCHILD(node);
nodelist := DBE_XMLDOM.GETCHILDNODES(node);
len := DBE_XMLDOM.GETLENGTH(nodelist);
RAISE NOTICE 'len : %', len;
END;
/
```

- **DBE\_XMLDOM.GETLOCALNAME**

Returns the local name of the given DOMAttr node. The prototype of the DBE\_XMLDOM.MAKENODE function is as follows:

```
DBE_XMLDOM.GETLOCALNAME(
 a IN DOMATTR)
RETURN VARCHAR2;
```

Returns the local name of the given DOMELEMENT node. The prototype of the DBE\_XMLDOM.MAKENODE function is as follows:

```
DBE_XMLDOM.GETLOCALNAME(
 elem IN DOMELEMENT)
RETURN VARCHAR2;
```

Returns the local name of the given DOMNode node. The prototype of the DBE\_XMLDOM.MAKENODE function is as follows:

```
DBE_XMLDOM.GETLOCALNAME(
 n IN DOMNODE,
 data OUT VARCHAR2);
```

**Table 10-347** DBE\_XMLDOM.GETLOCALNAME parameters

| Parameter | Description               |
|-----------|---------------------------|
| a         | Specified DOMAttr node    |
| elem      | Specified DOMELEMENT node |
| n         | Specified DOMNode node    |
| data      | Returned local name       |

**Example:**

--2. Use the createElement function to generate a DOMELEMENT node and obtain its local name.

```
DECLARE
 doc DBE_xmlDOM.domdocument;
 elem DBE_xmlDOM.domelement;
 value varchar2(10000);
BEGIN
 doc := DBE_xmlDOM.newdomdocument();
 elem := DBE_XMLDOM.createELEMENT(doc, 'root');
 value := DBE_XMLDOM.getlocalname(elem);
 DBE_output.print_line('value: ');
 DBE_output.print_line(value);
END;
/
```

--3. Convert a DOMELEMENT to a DOMNode node and obtain its local name.

```
DECLARE
 doc DBE_xmlDOM.domdocument;
 elem DBE_xmlDOM.domelement;
 node DBE_xmlDOM.domnode;
 value varchar2(100);
 buf varchar2(100);
BEGIN
 doc := DBE_xmlDOM.newdomdocument();
```

```

elem := DBE_XMLDOM.createELEMENT(doc, 'root');
node := DBE_xmlDOM.makenode(elem);
DBE_XMLDOM.getlocalname(node, buf);
DBE_output.print_line('buf: ');
DBE_output.print_line(buf);
END;
/

```

- **DBE\_XMLDOM.GETNAMEDITEM**

Returns the node specified by name. The prototype of the DBE\_XMLDOM.GETNAMEDITEM function is as follows:

```

DBE_XMLDOM.GETNAMEDITEM(
 nnm IN DOMNAMEDNODEMAP,
 name IN VARCHAR2)
RETURN DOMNODE;

```

Returns the node specified by name and namespace. The prototype of the DBE\_XMLDOM.GETNAMEDITEM function is as follows:

```

DBE_XMLDOM.GETNAMEDITEM(
 nnm IN DOMNAMEDNODEMAP,
 name IN VARCHAR2,
 ns IN VARCHAR2)
RETURN DOMNODE;

```

**Table 10-348** DBE\_XMLDOM.GETNAMEDITEM parameters

| Parameter | Description                          |
|-----------|--------------------------------------|
| nnm       | Specified DOMNamedNodeMap object.    |
| name      | Name of the element to be retrieved. |
| ns        | Namespace.                           |

 **NOTE**

1. The values of **name** and **nnm** can be null, but they are required arguments.
2. The default maximum length of **name** and **ns** is 32767. If the length exceeds 32767, an error is reported.
3. The values of **name** and **ns** can be of the int type and contain more than 127 bits.

**Example:**

```

-- 1. Return the node specified by name.
DECLARE
 doc DBE_XMLDOM.DOMDocument;
 elem DBE_XMLDOM.DOMELEMENT;
 map DBE_XMLDOM.DOMNAMEDNODEMAP;
 node DBE_XMLDOM.DOMNODE;
 node2 DBE_XMLDOM.DOMNODE;
 buf varchar2(1000);
 buf2 varchar2(1000);
BEGIN
 doc := dbe_xmlDOM.newdomdocument('<bookstore category="web" cover="paperback">
 <book category="cooking"><title lang="en">Everyday Italian</title>
 <author>Giada De Laurentiis</author><year>2005</year>
 <price>30.00</price></book></bookstore>');
 elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
 node := DBE_XMLDOM.MAKENODE(elem);
 map := DBE_XMLDOM.GETATTRIBUTES(node);
 node2:= DBE_XMLDOM.GETNAMEDITEM(map,'category');

```

```

DBE_XMLDOM.writeToBuffer(node2, buf2);
dbe_output.print_line(buf2);
END;
/

-- 2. Return the node specified by name and namespace.
DECLARE
doc DBE_XMLDOM.DOMDocument;
root DBE_XMLDOM.DOMELEMENT;
elem DBE_XMLDOM.DOMELEMENT;
map DBE_XMLDOM.DOMNAMEDNODEMAP;
node DBE_XMLDOM.DOMNODE;
buf varchar2(1000);
buf2 varchar2(1000);
BEGIN
doc := dbe_xmldom.newdomdocument('<h:table xmlns:h="http://www.w3.org/TR/html4/">
<h:tr h:id="10"><h:td >Apples</h:td>
<h:td>Bananas</h:td></h:tr></h:table>');
root := DBE_XMLDOM.getDocumentElement(doc);
node := DBE_XMLDOM.MAKENODE(root);
node := dbe_xmldom.GETFIRSTCHILD(node);
map := DBE_XMLDOM.GETATTRIBUTES(node);
node := DBE_XMLDOM.GETNAMEDITEM(map,'id','http://www.w3.org/TR/html4/');
DBE_XMLDOM.writeToBuffer(node, buf2);
dbe_output.print_line(buf2);
END;
/

```

- **DBE\_XMLDOM.GETNEXTSIBLING**

Returns the next node. The prototype of the DBE\_XMLDOM.GETNEXTSIBLING function is as follows:

```

DBE_XMLDOM.GETNEXTSIBLING(
n IN DOMNODE)
RETURN DOMNODE;

```

**Table 10-349** DBE\_XMLDOM.GETNEXTSIBLING parameters

| Parameter | Description            |
|-----------|------------------------|
| n         | Specified DOMNode node |

**Example:**

-- Obtain the first subnode after the DOC tree is converted into the DOMNode type, and obtain the name of the first subnode of the obtained first DOMNode node. Then, obtain the name of the next node through DBE\_XMLDOM.GETNEXTSIBLING.

```

DECLARE
doc dbe_xmldom.domdocument;
doc_node dbe_xmldom.domnode;
root_node dbe_xmldom.domnode;
inside_node dbe_xmldom.domnode;
node_name varchar2(1000);
node_type integer;
BEGIN
doc := dbe_xmldom.newdomdocument('<computer size="ITX">
<cpu>Ryzen 9 3950X</cpu>
<ram>32GBx2 DDR4 3200MHz</ram>
<motherboard>X570i</motherboard>
</computer>');
doc_node := DBE_XMLDOM.MAKENODE(doc);
root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
node_name := DBE_XMLDOM.GETNODENAME(root_node);
node_type := DBE_XMLDOM.GETNODETYPE(root_node);
dbe_output.print_line(node_name);
dbe_output.print_line(node_type);
inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);

```

```
node_name := DBE_XMLDOM.GETNODENAME(inside_node);
dbe_output.print_line(node_name);
inside_node := DBE_XMLDOM.GETNEXTSIBLING(inside_node);
node_name := DBE_XMLDOM.GETNODENAME(inside_node);
dbe_output.print_line(node_name);
END;
/
```

- **DBE\_XMLDOM.GETNODENAME**

Returns the name of a node. The prototype of the DBE\_XMLDOM.GETNODENAME function is as follows:

```
DBE_XMLDOM.GETNODENAME(
 n IN DOMNODE)
RETURN VARCHAR2;
```

**Table 10-350** DBE\_XMLDOM.GETNODENAME parameters

| Parameter | Description             |
|-----------|-------------------------|
| n         | Specified DOMNode node. |

**Example:**

```
-- Obtain the name of the specified DOMNode node from the DOC tree.
DECLARE
doc DBE_XMLDOM.DOMDocument;
root DBE_XMLDOM.DOMELEMENT;
root_node DBE_XMLDOM.DOMNode;
inside_node DBE_XMLDOM.DOMNode;
buf VARCHAR2(1000);
BEGIN
doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
<book category="cooking"><title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author><year>2005</year>
<price>30.00</price></book></bookstore>');
root := DBE_XMLDOM.getDocumentElement(doc);
root_node := DBE_XMLDOM.MAKENODE(root);
inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);
buf := DBE_XMLDOM.GETNODENAME(inside_node);
dbe_output.print_line(buf);
END;
/
```

- **DBE\_XMLDOM.GETNODETYPE**

Returns the type of a node. The prototype of the DBE\_XMLDOM.GETNODETYPE function is as follows:

```
DBE_XMLDOM.GETNODETYPE(
 n IN DOMNODE)
RETURN NUMBER;
```

**Table 10-351** DBE\_XMLDOM.GETNODETYPE parameters

| Parameter | Description            |
|-----------|------------------------|
| n         | Specified DOMNode node |

**Example:**

```
-- Obtain the type of the specified DOMNode node from the DOC tree.
DECLARE
doc DBE_XMLDOM.DOMDocument;
```

```

doc_node DBE_XMLDOM.DOMNode;
num number;
buf varchar2(1000);
BEGIN
 doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
 <book category="cooking"><title lang="en">Everyday Italian</title>
 <author>Giada De Laurentiis</author><year>2005</year>
 <price>30.00</price></book></bookstore>');
 doc_node := DBE_XMLDOM.makeNode(doc);
 num := DBE_XMLDOM.GETNODETYPE(doc_node);
 dbe_output.print_line(num);
 buf := DBE_XMLDOM.GETNODENAME(doc_node);
 dbe_output.print_line(buf);
END;
/

```

- DBE\_XMLDOM.GETNODEVALUE

Returns the value of a DOMNode node. The prototype of the DBE\_XMLDOM.GETNODEVALUE function is as follows:

```

DBE_XMLDOM.GETNODEVALUE(
 n IN DOMNODE)
RETURN VARCHAR2;

```

**Table 10-352** DBE\_XMLDOM.GETNODEVALUE parameters

| Parameter | Description              |
|-----------|--------------------------|
| n         | Specified DOMNode object |

**Example:**

```

-- Convert a DOMText node to a DOMNode node and obtain the value of the node.
DECLARE
 buf VARCHAR2(1000);
 doc DBE_XMLDOM.DOMDocument;
 text DBE_XMLDOM.DOMText;
 elem2 DBE_XMLDOM.DOMEElement;
 node DBE_XMLDOM.DOMNode;
begin
 doc := DBE_XMLDOM.NEWDOMDOCUMENT();
 text := DBE_XMLDOM.createTextNode(doc, 'aaa');
 DBE_XMLDOM.SETNODEVALUE(DBE_XMLDOM.makeNode(text), 'ccc');
 buf := DBE_XMLDOM.GETNODEVALUE(DBE_XMLDOM.makeNode(text));
 DBE_OUTPUT.print_line(buf);
end;
/

```

- DBE\_XMLDOM.GETPARENTNODE

Returns the parent node of the given DOMNode node. The prototype of the DBE\_XMLDOM.GETPARENTNODE function is as follows:

```

DBE_XMLDOM.GETPARENTNODE(
 n IN DOMNODE)
RETURN DOMNODE;

```

**Table 10-353** DBE\_XMLDOM.GETPARENTNODE parameters

| Parameter | Description               |
|-----------|---------------------------|
| n         | Specified DOMNode object. |

**Example:**

```
-- Add a node to the DOC tree and obtain the name of its parent node.
DECLARE
 doc DBE_XMLDOM.DOMDocument;
 doc1 DBE_XMLDOM.DOMDocument;
 root DBE_XMLDOM.DOMELEMENT;
 child1 DBE_XMLDOM.DOMELEMENT;
 child2 DBE_XMLDOM.DOMELEMENT;
 attr DBE_XMLDOM.DOMAttr;
 text DBE_XMLDOM.DOMTEXT;
 node DBE_XMLDOM.DOMNode;
 parent DBE_XMLDOM.DOMNode;
 buf varchar2(1000);
BEGIN
 doc := DBE_XMLDOM.newDOMDocument();
 root := DBE_XMLDOM.createElement(doc, 'root');
 node := DBE_XMLDOM.appendChild(DBE_xmldom.makeNode(doc),DBE_xmldom.makeNode(root));
 child1 := DBE_XMLDOM.createElement(doc, 'child1');
 node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(root),
DBE_XMLDOM.makeNode(child1));
 child2 := DBE_XMLDOM.createElement(doc, 'child2');
 node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(child1),
DBE_XMLDOM.makeNode(child2));
 parent := DBE_XMLDOM.GETPARENTNODE(DBE_XMLDOM.makeNode(child2));
 buf := DBE_XMLDOM.GETNODENAME(parent);
 DBE_OUTPUT.print_line(buf);
END;
/
```

- **DBE\_XMLDOM.GETTAGNAME**

Returns the tag name of the specified DOMELEMENT node. The prototype of the DBE\_XMLDOM.GETTAGNAME function is as follows:

```
DBE_XMLDOM.GETTAGNAME(
 elem IN DOMELEMENT)
RETURN VARCHAR2;
```

**Table 10-354** DBE\_XMLDOM.GETTAGNAME parameters

| Parameter | Description               |
|-----------|---------------------------|
| elem      | Specified DOMELEMENT node |

**Example:**

```
-- Obtain the tag name of a DOMELEMENT node.
DECLARE
 doc dbe_xmldom.domdocument;
 elem dbe_xmldom.domelement;
 buffer varchar2(1010);
 value varchar(1000);
BEGIN
 doc := dbe_xmldom.newDOMDocument();
 elem := DBE_XMLDOM.CREATEELEMENT(DBE_XMLDOM.NEWDOMDOCUMENT(), 'root');
 value := DBE_XMLDOM.gettagname(elem);
 dbe_output.print_line('value: ');
 dbe_output.print_line(value);
 dbe_xmldom.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
END;
/
```

- **DBE\_XMLDOM.HASCHILDNODES**

Checks whether the DOMNode object has any subnode. The prototype of the DBE\_XMLDOM.HASCHILDNODES function is as follows:

```
DBE_XMLDOM.HASCHILDNODES(
 n IN DOMNODE)
RETURN BOOLEAN;
```

**Table 10-355** DBE\_XMLDOM.HASCHILDNODES parameters

| Parameter | Description              |
|-----------|--------------------------|
| n         | Specified DOMNode object |

**Example:**

```
-- Create a node named child1, mount it to the DOC tree, and add a node to child1. Then, check
whether the child1 node has any subnode.
DECLARE
 doc DBE_XMLDOM.DOMDocument;
 doc1 DBE_XMLDOM.DOMDocument;
 root DBE_XMLDOM.DOMELEMENT;
 child1 DBE_XMLDOM.DOMELEMENT;
 child2 DBE_XMLDOM.DOMELEMENT;
 attr DBE_XMLDOM.DOMAttr;
 text DBE_XMLDOM.DOMTEXT;
 node DBE_XMLDOM.DOMNODE;
 buf varchar2(1000);
BEGIN
 doc := DBE_XMLDOM.newDOMDocument();
 root := DBE_XMLDOM.createElement(doc, 'root');
 node := DBE_XMLDOM.appendChild(DBE_xmldom.makeNode(doc),DBE_xmldom.makeNode(root));
 child1 := DBE_XMLDOM.createElement(doc, 'child1');
 node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(root),
DBE_XMLDOM.makeNode(child1));
 child2 := DBE_XMLDOM.createElement(doc, 'child2');
 node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(child1),
DBE_XMLDOM.makeNode(child2));
 IF DBE_XMLDOM.HASCHILDNODES(DBE_XMLDOM.makeNode(child1)) THEN
 DBE_OUTPUT.print_line('HAS CHILD NODES');
 ELSE
 DBE_OUTPUT.print_line('NOT HAS CHILD NODES ');
 END IF;
END;
```

- **DBE\_XMLDOM.IMPORTNODE**

Copies a node to another node and mounts the copied node to a specified document. If the type of the copied node does not belong to the 12 types specified by constants of XML DOM, an exception indicating that the type is not supported is thrown. The prototype of the DBE\_XMLDOM.IMPORTNODE function is as follows:

```
DBE_XMLDOM.IMPORTNODE(
 doc IN DOMDOCUMENT,
 importedNode IN DOMNODE,
 deep IN BOOLEAN)
RETURN DOMNODE;
```

**Table 10-356** DBE\_XMLDOM.IMPORTNODE parameters

| Parameter | Description                            |
|-----------|----------------------------------------|
| doc       | Document to which the node is mounted. |

| Parameter    | Description                                                                                                                                                                                                                                   |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| importedNode | Node to be imported.                                                                                                                                                                                                                          |
| deep         | Specifies whether to perform recursive import. <ul style="list-style-type: none"> <li>If the value is <b>TRUE</b>, the node and all its subnodes are imported.</li> <li>If the value is <b>FALSE</b>, the node itself is imported.</li> </ul> |

**Example:**

-- Obtain the **root2\_node** node in the **DOC2** tree, copy it, and mount it to the **DOC** tree.

```
DECLARE
 doc dbe_xmlDOM.domdocument;
 doc2 dbe_xmlDOM.domdocument;
 doc_node dbe_xmlDOM.domnode;
 doc2_node dbe_xmlDOM.domnode;
 root_node dbe_xmlDOM.domnode;
 root2_node dbe_xmlDOM.domnode;
 import_node dbe_xmlDOM.domnode;
 result_node dbe_xmlDOM.domnode;
 buffer varchar2(1010);
BEGIN
 doc := dbe_xmlDOM.newDOMdocument('<bookstore category="web" cover="paperback">
 <book category="cooking"><title lang="en">Everyday Italian</title>
 <author>Giada De Laurentiis</author><year>2005</year>
 <price>30.00</price></book></bookstore>');
 doc2 := dbe_xmlDOM.newDOMdocument('<case>LIANLI TU150</case>');
 doc_node := DBE_XMLDOM.MAKENODE(doc);
 doc2_node := DBE_XMLDOM.MAKENODE(doc2);
 root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
 root2_node := DBE_XMLDOM.GETFIRSTCHILD(doc2_node);
 DBE_XMLDOM.WRITETOBUFFER(doc, buffer);
 dbe_output.print_line(buffer);
 import_node := DBE_XMLDOM.IMPORTNODE(doc, root2_node, TRUE);
 result_node := DBE_XMLDOM.APPENDCHILD(root_node, import_node);
 DBE_XMLDOM.WRITETOBUFFER(doc, buffer);
 dbe_output.print_line(buffer);
END;
/
```

- **DBE\_XMLDOM.ISNULL**

Checks whether the given DOMAttr node is null. If so, **TRUE** is returned. If not, **FALSE** is returned. The prototype of the DBE\_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
 a IN DOMATTR)
RETURN BOOLEAN;
```

Checks whether the given DOMDocument node is null. If so, **TRUE** is returned. If not, **FALSE** is returned. The prototype of the DBE\_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
 doc IN DOMDOCUMENT)
RETURN BOOLEAN;
```

Checks whether the given DOMELEMENT node is null. If so, **TRUE** is returned. If not, **FALSE** is returned. The prototype of the DBE\_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
 elem IN DOMELEMENT)
RETURN BOOLEAN;
```

Checks whether the given DOMNamedNodeMap node is null. If so, **TRUE** is returned. If not, **FALSE** is returned. The prototype of the DBE\_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
 nnm IN DOMNAMEDNODEMAP)
RETURN BOOLEAN;
```

Checks whether the given DOMNode node is null. If so, **TRUE** is returned. If not, **FALSE** is returned. The prototype of the DBE\_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
 n IN DOMNODE)
RETURN BOOLEAN;
```

Checks whether the given DOMNodeList node is null. If so, **TRUE** is returned. If not, **FALSE** is returned. The prototype of the DBE\_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
 nl IN DOMNODELIST)
RETURN BOOLEAN;
```

Checks whether the given DOMText node is null. If so, **TRUE** is returned. If not, **FALSE** is returned. The prototype of the DBE\_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
 t IN DOMTEXT)
RETURN BOOLEAN;
```

**Table 10-357** DBE\_XMLDOM.ISNULL parameters

| Parameter | Description                     |
|-----------|---------------------------------|
| a         | Specified DOMAttr node.         |
| doc       | Specified DOMDocument node.     |
| elem      | Specified DOMELEMENT node.      |
| nnm       | Specified DOMNamedNodeMap node. |
| n         | Specified DOMNode node.         |
| nl        | Specified DOMNodeList node.     |
| t         | Specified DOMText node.         |

 **NOTE**

Due to the implementation difference of DBE\_XMLDOM.FREEDOCUMENT, an error is reported when the DBE\_XMLDOM.ISNULL API calls the freed DOMDocument node.

Example:

```
--2. Declare (but not initialize) a DOMELEMENT node and check whether the node is empty.
DECLARE
docelem DBE_XMLDOM.DOMELEMENT;
```

```
BEGIN
 if DBE_XMLDOM.ISNULL(docelem) then
 DBE_OUTPUT.print_line('null');
 else
 DBE_OUTPUT.print_line('not null');
 end if;
END;
/
```

- **DBE\_XMLDOM.ITEM**

Returns the element corresponding to the index in a list based on the index. The prototype of the DBE\_XMLDOM.ITEM function is as follows:

```
DBE_XMLDOM.ITEM(
 nl IN DOMNODELIST,
 index IN NUMBER)
RETURN DOMNODE;
```

Returns the element corresponding to the index in a map based on the index. The prototype of the DBE\_XMLDOM.ITEM function is as follows:

```
DBE_XMLDOM.ITEM(
 nnm IN DOMNAMEDNODEMAP,
 index IN NUMBER)
RETURN DOMNODE;
```

**Table 10-358** DBE\_XMLDOM.ITEM parameters

| Parameter | Description                          |
|-----------|--------------------------------------|
| nl        | Specified DOMNodeList object         |
| nnm       | Specified DOMNamedNodeMap object     |
| index     | Index of the element to be retrieved |

 **NOTE**

For improper input parameters such as Boolean and CLOB, the item function of the map type points to the value of the first index by default.

**Example:**

--1. Return the element corresponding to the index in a map based on the index.

```
DECLARE
 doc DBE_XMLDOM.DOMDocument;
 elem DBE_XMLDOM.DOMELEMENT;
 map DBE_XMLDOM.DOMNAMEDNODEMAP;
 node DBE_XMLDOM.DOMNODE;
 node2 DBE_XMLDOM.DOMNODE;
 buf varchar2(1000);
BEGIN
 doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback"><book
category="cooking">
 <title lang="en">Everyday Italian</title><author>Giada De Laurentiis</author>
 <year>2005</year><price>30.00</price></book></bookstore>');
 elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
 node := DBE_XMLDOM.MAKENODE(elem);
 map := DBE_XMLDOM.GETATTRIBUTES(DBE_XMLDOM.getFirstChild(node));
 node2:= DBE_XMLDOM.item(map,0);
 DBE_XMLDOM.writeToBuffer(node2, buf);
 dbe_output.print_line(buf);
 dbe_xmldom.freedocument(doc);
 RAISE NOTICE '%', buf;
END;
```

```

/
--2. Return the element corresponding to the index in a list based on the index.
DECLARE
 doc dbe_xmlDOM.DOMDOCUMENT;
 node dbe_xmlDOM.DOMNODE;
 node1 dbe_xmlDOM.DOMNODE;
 nodelist DBE_XMLDOM.DOMNODELIST;
 len INTEGER;
 buffer1 VARCHAR2(1010);
BEGIN
 doc := dbe_xmlDOM.newDOMDOCUMENT('<bookstore category="web" cover="paperback"><book
category="cooking">
 <title lang="en">Everyday Italian</title><author>Giada De Laurentiis</author>
 <year>2005</year><price>30.00</price></book></bookstore>');
 node := dbe_xmlDOM.makeNode(doc);
 node := dbe_xmlDOM.GETFIRSTCHILD(node);
 node := dbe_xmlDOM.GETFIRSTCHILD(node);
 nodelist := DBE_XMLDOM.GETCHILDNODES(node);
 len := DBE_XMLDOM.GETLENGTH(nodelist);
 RAISE NOTICE 'len : %', len;
 node1 := DBE_XMLDOM.ITEM(nodelist, 0);
 IF DBE_XMLDOM.ISNULL(node1) THEN
 dbe_output.print_line('IS NULL');
 ELSE
 dbe_output.print_line('NOT NULL');
 END IF;
 dbe_xmlDOM.writetobuffer(node1, buffer1);
 dbe_output.print_line('buffer1: ');
 dbe_output.print_line(buffer1);
END;
/

```

- **DBE\_XMLDOM.MAKEELEMENT**

Returns the DOMElement object after conversion. The prototype of the DBE\_XMLDOM.MAKEELEMENT function is as follows:

```

DBE_XMLDOM.MAKEELEMENT(
 n IN DOMNODE)
RETURN DOMELEMENT;

```

**Table 10-359** DBE\_XMLDOM.MAKEELEMENT parameters

Parameter	Description
n	Specified DOMNode object

**Example:**

-- Forcibly convert the DOMNode node converted from the DOMElement type back to the DOMElement type.

```

DECLARE
 buf VARCHAR2(1000);
 doc DBE_XMLDOM.DOMDOCUMENT;
 elem DBE_XMLDOM.DOMELEMENT;
 elem2 DBE_XMLDOM.DOMELEMENT;
 node DBE_XMLDOM.DOMNODE;
BEGIN
 doc := DBE_XMLDOM.NEWDOMDOCUMENT();
 elem := DBE_XMLDOM.createElement(doc, 'aaa');
 node := DBE_XMLDOM.makeNode(elem);
 elem2 := DBE_XMLDOM.makeElement(node);
 buf := DBE_XMLDOM.GETNODENAME(DBE_XMLDOM.makeNode(elem2));
 DBE_OUTPUT.print_line(buf);
END;
/

```

- DBE\_XMLDOM.MAKENODE

Forcibly converts a specified DOMAttr node to a DOMNode node and returns the DOMNode node. The prototype of the DBE\_XMLDOM.MAKENODE function is as follows:

```
DBE_XMLDOM.MAKENODE(
 a IN DOMATTR)
RETURN DOMNODE;
```

Forcibly converts a specified DOMDocument node to a DOMNode node and returns the DOMNode node. The prototype of the DBE\_XMLDOM.MAKENODE function is as follows:

```
DBE_XMLDOM.MAKENODE(
 doc IN DOMDOCUMENT)
RETURN DOMNODE;
```

Forcibly converts a specified DOMELEMENT node to a DOMNode node and returns the DOMNode node. The prototype of the DBE\_XMLDOM.MAKENODE function is as follows:

```
DBE_XMLDOM.MAKENODE(
 elem IN DOMELEMENT)
RETURN DOMNODE;
```

Forcibly converts a specified DOMText node to a DOMNode node and returns the DOMNode node. The prototype of the DBE\_XMLDOM.MAKENODE function is as follows:

```
DBE_XMLDOM.MAKENODE(
 t IN DOMTEXT)
RETURN DOMNODE;
```

**Table 10-360** DBE\_XMLDOM.MAKENODE parameters

Parameter	Description
a	Specified DOMAttr node
doc	Specified DOMDocument node
elem	Specified DOMELEMENT node
t	Specified DOMText node

 **NOTE**

Due to syntax restrictions, when DBE\_XMLDOM.MAKENODE is used as the return value of a function, it cannot be directly implemented by running the following command:

```
return DBE_XMLDOM.MAKENODE(doc);
```

You are advised to run the following command:

```
tmp_node := DBE_XMLDOM.MAKENODE(doc);
return tmp_node;
```

**Example:**

```
--4. Declare (but not initialize) a DOMText variable, and use it as the input parameter of MAKENODE.
DECLARE
text DBE_XMLDOM.DOMTEXT;
buf VARCHAR2(1000);
```

```

dom_node DBE_XMLDOM.DOMNODE;
BEGIN
 dom_node := DBE_XMLDOM.makeNode(text);
 buf := DBE_XMLDOM.GETNODENAME(dom_node);
 DBE_OUTPUT.print_line(buf);
END;
/

```

- **DBE\_XMLDOM.NEWDOMDOCUMENT**

Returns a new DOMDocument object. The prototype of the DBE\_XMLDOM.NEWDOMDOCUMENT function is as follows:

```

DBE_XMLDOM.NEWDOMDOCUMENT
RETURN DOMDOCUMENT;

```

Returns a new DOMDocument instance object created from the specified XMLType type. The prototype of the DBE\_XMLDOM.NEWDOMDOCUMENT function is as follows:

```

DBE_XMLDOM.NEWDOMDOCUMENT(
 xmlDoc IN SYS.XMLTYPE)
RETURN DOMDOCUMENT;

```

Returns a new DOMDocument instance object created from the specified CLOB type. The prototype of the DBE\_XMLDOM.NEWDOMDOCUMENT function is as follows:

```

DBE_XMLDOM.NEWDOMDOCUMENT(
 cl IN CLOB)
RETURN DOMDOCUMENT;

```

**Table 10-361** DBE\_XMLDOM.NEWDOMDOCUMENT parameters

Parameter	Description
xmlDoc	Specified XMLType type
cl	Specified CLOB type

 **NOTE**

- The size of the input parameter must be less than 1 GB.
- Currently, external DTD parsing is not supported.
- The document created by NEWDOMDOCUMENT uses the UTF-8 character set by default.
- Each document parsed from the same XMLType instance is independent, and the modification of the document does not affect the XMLType.
- For details about the differences between our database ORAnd database ORA, see [DBE\\_XMLPARSER.PARSECLOB](#).

**Example:**

```

--1. Return a new DOMDocument object.
DECLARE
 doc dbe_xmlDOM.domdocument;
 buffer varchar2(1010);
BEGIN
 doc := dbe_xmlDOM.newdomdocument();
 dbe_xmlDOM.setdoctype(doc, 'note', 'sysid', 'pubid');
 dbe_xmlDOM.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
 dbe_xmlDOM.freedocument(doc);
END;
/

```

```
--2. Return a new DOMDocument instance object created from the specified CLOB type.
DECLARE
 doc dbexml.domdocument;
 buffer varchar2(1010);
BEGIN
 doc := dbexml.newdomdocument('<?xml version="1.0"?>
 <note><to>test</to><from>Jani</from><heading>Reminder</heading>
 <body>Don't forget me this weekend!</body></note>');
 dbexml.writetobuffer(doc, buffer);
 db_output.print_line('buffer: ');
 db_output.print_line(buffer);
 dbexml.freedocument(doc);
END;
/

--3. Return a new DOMDocument instance object created from the specified XMLType type.
DECLARE
 doc dbexml.domdocument;
 xt xmltype;
 buffer varchar2(1010);
BEGIN
 xt := xmltype('<h:data xmlns:h="http://www.w3.org/TR/html4/">
 <h:da1 len="10">test namespace</h:da1>
 <h:da1>bbbbbbbbbb</h:da1>
 </h:data>');
 doc := dbexml.newdomdocument(xt);
 dbexml.writetobuffer(doc, buffer);
 db_output.print_line('buffer: ');
 db_output.print_line(buffer);
 dbexml.freedocument(doc);
END;
/
```

- **DBE\_XMLDOM.SETATTRIBUTE**

Sets the value of the DOMELEMENT attribute by name. The prototype of the DBE\_XMLDOM.SETATTRIBUTE function is as follows:

```
DBE_XMLDOM.SETATTRIBUTE(
 elem IN DOMELEMENT,
 name IN VARCHAR2,
 value IN VARCHAR2);
```

Sets the attribute values of a DOMELEMENT object by name and namespace URI. The prototype of the DBE\_XMLDOM.SETATTRIBUTE function is as follows:

```
DBE_XMLDOM.SETATTRIBUTE(
 elem IN DOMELEMENT,
 name IN VARCHAR2,
 value IN VARCHAR2,
 ns IN VARCHAR2);
```

**Table 10-362** DBE\_XMLDOM.SETATTRIBUTE parameters

Parameter	Description
elem	Specified DOMELEMENT node
name	Attribute name
value	Attribute value
ns	Namespace

 **NOTE**

Multiple attributes can be added through the DBE\_XMLDOM.SETATTRIBUTE API. The attribute name cannot be null, and attributes with the same name cannot exist in the same DOMELEMENT node. If you want to add attributes with the same name, you should explicitly set a namespace for each attribute with the same name, but you are advised not to perform such operations. If an attribute exists in a namespace, the specified namespace must be displayed when you modify the attribute. Otherwise, the attribute with the same name is added.

**Example:**

```
--1. Set the value of the DOMELEMENT attribute by name.
DECLARE
 doc dbe_xmldom.domdocument;
 elem dbe_xmldom.domelement;
 docnode DBE_XMLDOM.DOMNode;
 buffer varchar2(1010);
 value varchar(1000);
BEGIN
 doc := dbe_xmldom.newDOMDocument();
 elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
 DBE_XMLDOM.setattribute(elem, 'len', '50cm');
 docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
 dbe_xmldom.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
END;
/

--2. Set the attribute values of a DOMELEMENT object by name and namespace URI.
DECLARE
 doc dbe_xmldom.domdocument;
 elem dbe_xmldom.domelement;
 docnode DBE_XMLDOM.DOMNode;
 buffer varchar2(1010);
 value varchar(1000);
begin
 doc := dbe_xmldom.newDOMDocument();
 elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
 DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
 docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
 dbe_xmldom.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
END;
/

--3. Change the values of the DOMELEMENT attributes by name.
DECLARE
 doc dbe_xmldom.domdocument;
 elem dbe_xmldom.domelement;
 docnode DBE_XMLDOM.DOMNode;
 buffer varchar2(1010);
 value varchar(1000);
BEGIN
 doc := dbe_xmldom.newDOMDocument();
 elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
 DBE_XMLDOM.setattribute(elem, 'len', '50cm');
 DBE_XMLDOM.setattribute(elem, 'len', '55cm');
 docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
 dbe_xmldom.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
END;
/
```

```
--4. Change the values of the DOMElement attributes by name and namespace URI.
DECLARE
 doc dbe_xmlDOM.domdocument;
 elem dbe_xmlDOM.domelement;
 docnode DBE_XMLDOM.DOMNode;
 buffer varchar2(1010);
 value varchar(1000);
begin
 doc := dbe_xmlDOM.newDOMDocument();
 elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
 DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
 DBE_XMLDOM.setattribute(elem, 'len', '55cm', 'www.huawei.com');
 docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
 dbe_xmlDOM.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
END;
/
```

- DBE\_XMLDOM.SETCHARSET

Sets the character set for a DOMDocument object. The prototype of the DBE\_XMLDOM.SETCHARSET function is as follows:

```
DBE_XMLDOM.SETCHARSET(
 doc IN DOMDocument,
 charset IN VARCHAR2);
```

**Table 10-363** DBE\_XMLDOM.SETCHARSET parameters

Parameter	Description
doc	Specified DOMDocument node
charset	Character set

 **NOTE**

- The value of **charset** contains a maximum of 60 bytes.
- Currently, the following character sets are supported: UTF-8, UTF-16, UCS-4, UCS-2, ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, ISO-2022-JP, Shift\_JIS, EUC-JP and ASCII. If you enter other character sets, an error is reported or garbled characters may be displayed.

**Example:**

```
-- Set the UTF-16 character set for the DOC tree and print the DOC tree to the buffer.
DECLARE
 doc dbe_xmlDOM.domdocument;
 buffer varchar2(1010);
BEGIN
 doc := dbe_xmlDOM.newdomdocument('<?xml version="1.0"?>
<!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
<note><to>test</to><from>Jani</from><heading>Reminder</heading>
<body>Don"t forget me this weekend!</body></note>');
 dbe_xmlDOM.setcharset(doc, 'utf-16');
 dbe_xmlDOM.writetobuffer(doc, buffer);
 dbe_output.print_line('buffer: ');
 dbe_output.print_line(buffer);
 dbe_xmlDOM.freedocument(doc);
END;
/
```

- DBE\_XMLDOM.SETDOCTYPE

Sets the external DTD of a DOMDocument object. The prototype of the DBE\_XMLDOM.SETDOCTYPE function is as follows:

```
DBE_XMLDOM.SETDOCTYPE(
doc IN DOMDocument,
name IN VARCHAR2,
sysid IN VARCHAR2,
pubid IN VARCHAR2);
```

**Table 10-364** DBE\_XMLDOM.SETDOCTYPE parameters

Parameter	Description
doc	Specified DOMDocument node
name	Name of the DOCTYPE to be initialized
sysid	ID of the system whose DOCTYPE needs to be initialized
pubid	Public ID of the DOCTYPE to be initialized

 **NOTE**

The total length of **name**, **sysid**, and **pubid** cannot exceed 32500 bytes.

**Example:**

-- After the initial system ID, public ID, and name are set for the external DTD of the DOMDocument, the DOC tree modified each time is output to the buffer.

```
DECLARE
doc dbe_xmldom.domdocument;
buffer varchar2(1010);
begin
doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
<!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
<note><to>test</to><from>Jani</from><heading>Reminder</heading>
<body>Don''t forget me this weekend!</body></note>');
dbe_xmldom.setdoctype(doc, 'note', 'sysid', 'pubid');
dbe_xmldom.writetobuffer(doc, buffer);
dbe_output.print_line('buffer: ');
dbe_output.print_line(buffer);
dbe_output.print_line('-----');
dbe_xmldom.setdoctype(doc, 'note', NULL, '');
dbe_xmldom.setdoctype(doc, 'note', NULL, '');
dbe_xmldom.writetobuffer(doc, buffer);
dbe_output.print_line('buffer: ');
dbe_output.print_line(buffer);
dbe_xmldom.freedocument(doc);
END;
```

- DBE\_XMLDOM.SETNODEVALUE

Sets the value of a node in the DOMNode object. The prototype of the DBE\_XMLDOM.SETNODEVALUE function is as follows:

```
DBE_XMLDOM.SETNODEVALUE(
n IN DOMNODE,
nodeValue IN VARCHAR2);
```

**Table 10-365** DBE\_XMLDOM.SETNODEVALUE parameters

Parameter	Description
n	Specified DOMNode object
nodeValue	String to be set in the DOMNode object

 **NOTE**

1. You can enter an empty string or null value for **nodeValue**, but the node value will not be changed.
2. Currently, **nodeValue** does not support the escape character '&'. If the character string contains the escape character, the node value will be cleared.
3. The default maximum length of **nodeValue** is restricted by the VARCHAR2 type and is 32767 bytes. If the length exceeds 32767 bytes, an exception is thrown.

**Example:**

-- After setting a node value different from the initial value for the DOMNode node that is converted from DOMText, obtain and output the node value.

```
DECLARE
 buf VARCHAR2(1000);
 doc DBE_XMLDOM.DOMDocument;
 text DBE_XMLDOM.DOMText;
 elem2 DBE_XMLDOM.DOMELEMENT;
 node DBE_XMLDOM.DOMNode;
BEGIN
 doc := DBE_XMLDOM.NEWDOMDOCUMENT();
 text := DBE_XMLDOM.createTextNode(doc, 'aaa');
 DBE_XMLDOM.SETNODEVALUE(DBE_XMLDOM.makeNode(text), 'ccc');
 buf := DBE_XMLDOM.GETNODEVALUE(DBE_XMLDOM.makeNode(text));
 DBE_OUTPUT.print_line(buf);
END;
/
```

- **DBE\_XMLDOM.WRITETOBUFFER**

Writes an XML node to a specified buffer using the database character set.

The prototype of the DBE\_XMLDOM.WRITETOBUFFER function is as follows:

```
DBE_XMLDOM.WRITETOBUFFER(
 doc IN DOMDOCUMENT,
 buffer INOUT VARCHAR2);
```

Writes an XML document to a specified buffer using the database character set. The prototype of the DBE\_XMLDOM.WRITETOBUFFER function is as follows:

```
DBE_XMLDOM.WRITETOBUFFER(
 n IN DOMNODE,
 buffer INOUT VARCHAR2);
```

**Table 10-366** DBE\_XMLDOM.WRITETOBUFFER parameters

Parameter	Description
doc	Specified DOMDocument node
buffer	Buffer for the write operation
n	Specified DOMNode node

 **NOTE**

- The buffer to which the writetobuffer function writes is less than 1 GB.
- This function adds content such as indentation to format the output. The output document will contain the XML declaration version and encoding.
- By default, XML files are output in the UTF-8 character set.

**Example:**

```
--1. Enter a parameter of the DOMNode type.
DECLARE
 doc dbe_xmlDOM.domdocument;
 elem DBE_XMLDOM.DOMELEMENT;
 buf varchar2(1000);
BEGIN
 doc := dbe_xmlDOM.newdomdocument();
 elem := dbe_xmlDOM.createelement(doc,'elem');
 DBE_XMLDOM.WRITETOBUFFER(dbe_xmlDOM.makenode(elem), buf);
 DBE_OUTPUT.print_line(buf);
END;
/

--2. Enter a parameter of the DOMDocument type.
DECLARE
 doc DBE_XMLDOM.DOMDocument;
 buf VARCHAR2(1000);
BEGIN
 doc := dbe_xmlDOM.newdomdocument('<?xml version="1.0"?>
 <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
 <note><to>test</to><from>Jani</from><heading>Reminder</heading>
 <body>Don''t forget me this weekend!</body></note>');
 DBE_XMLDOM.WRITETOBUFFER(doc, buf);
 DBE_OUTPUT.print_line('doc: ');
 DBE_OUTPUT.print_line(buf);
 DBE_XMLDOM.FREEDOCUMENT(doc);
END;
/
```

- **DBE\_XMLDOM.WRITETOCLOB**

Writes an XML node to a specified CLOB using the database character set. The prototype of the DBE\_XMLDOM.WRITETOCLOB function is as follows:

```
DBE_XMLDOM.WRITETOCLOB(
 doc IN DOMDOCUMENT,
 cl INOUT CLOB);
```

Writes an XML node to a specified CLOB using the database character set. The prototype of the DBE\_XMLDOM.WRITETOCLOB function is as follows:

```
DBE_XMLDOM.WRITETOCLOB(
 n IN DOMNODE,
 cl INOUT CLOB);
```

**Table 10-367** DBE\_XMLDOM.WRITETOCLOB parameters

Parameter	Description
doc	Specified DOMDocument node
cl	CLOB to be written
n	Specified DOMNode node

 NOTE

- The **doc** parameter is an input parameter. The CLOB to which the writetoclob function writes is less than 1 GB.
- This function adds content such as indentation to format the output. The output document will contain the XML declaration version and encoding.
- By default, XML files are output in the UTF-8 character set.

## Example:

--1. Enter a parameter of the DOMNode type.

```
DECLARE
 CL CLOB;
 N DBE_XMLDOM.DOMNODE;
BEGIN
 DBE_XMLDOM.WRITETOCLOB(N, CL);
 DBE_OUTPUT.PRINT_LINE(CL);
END;
/
```

--2. Enter a parameter of the DOMDocument type.

```
DECLARE
 doc dbe_xmldom.domdocument;
 mclob clob;
BEGIN
 doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
 <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
 <note><to>test</to><from>Jani</from><heading>Reminder</heading>
 <body>Don't forget me this weekend!</body></note>');
 dbe_xmldom.writetoclob(doc, mclob);
 dbe_output.print_line('mclob: ');
 dbe_output.print_line(mclob);
 dbe_xmldom.freedocument(doc);
END;
/
```

- DBE\_XMLDOM.WRITETOFILE

Writes an XML node to a specified file using the database character set. The prototype of the DBE\_XMLDOM.WRITETOFILE function is as follows:

```
DBE_XMLDOM.WRITETOCLOB(
 doc IN DOMDOCUMENT,
 fileName IN VARCHAR2);
```

Writes an XML node to a specified file using the database character set. The prototype of the DBE\_XMLDOM.WRITETOFILE function is as follows:

```
DBE_XMLDOM.WRITETOCLOB(
 n IN DOMNODE,
 fileName IN VARCHAR2);
```

Writes an XML document to a specified file using the specified character set.

The prototype of the DBE\_XMLDOM.WRITETOFILE function is as follows:

```
DBE_XMLDOM.WRITETOCLOB(
 doc IN DOMDOCUMENT,
 fileName IN VARCHAR2,
 charset IN VARCHAR2);
```

Writes an XML document to a specified file using the specified character set.

The prototype of the DBE\_XMLDOM.WRITETOFILE function is as follows:

```
DBE_XMLDOM.WRITETOCLOB(
 n IN DOMNODE,
 fileName IN VARCHAR2,
 charset IN VARCHAR2);
```

**Table 10-368** DBE\_XMLDOM.WRITETOFILE parameters

Parameter	Description
doc	Specified DOMDocument node
fileName	File to be written
n	Specified DOMNode node
charset	Specified character set

**NOTE**

- The **doc** parameter is an input parameter. The value of **filename** can contain a maximum of 255 bytes, and the value of **charset** can contain a maximum of 60 bytes. For details about the character sets supported by **charset**, see the [DBE\\_XMLDOM.SETCHARSET](#) API.
- This function adds content such as indentation to format the output. The output document will contain the XML declaration version and encoding.
- If `newdomdocument()` is used to create a document without parameters, no error is reported when **charset** is not specified. The UTF-8 character set is used by default.
- The **filename** must be in the path created in **pg\_directory**. The backslash (\) in the filename will be converted to a slash (/). Only one slash (/) is allowed. The file name must be in the **pg\_directory\_name/file\_name.xml** format. The output file must be in the XML format.
- When the GUC parameter **safe\_data\_path** is enabled, you can only use the advanced package to read and write files in the file path specified by **safe\_data\_path**.
- Before creating a directory, ensure that the directory exists in the OS and the user has the read and write permissions on the directory. For details about how to create a directory, see [CREATE DIRECTORY](#).

**Example:**

-- Before creating a directory, ensure that the directory exists in the OS and the user has the read and write permissions on the directory.

create directory dir as '/tmp';

--1. Write an XML node to a specified file using the database character set.

```
DECLARE
 FPATH VARCHAR2(1000);
 DOC DBE_XMLDOM.DOMDOCUMENT;
BEGIN
 DOC := DBE_XMLDOM.NEWDOMDOCUMENT('<ROOT>
 <A ATTR1="A_VALUE">
 <ACHILD>ACHILD TXT</ACHILD>

 B TXT
 <C/>
 </ROOT>');
 FPATH := 'dir/simplexml.xml';
 DBE_XMLDOM.WRITETOFILE(DOC, FPATH);
END;
/
```

--2. Write an XML document to a specified file using the specified character set.

```
DECLARE
 SRC VARCHAR(1000);
 FPATH VARCHAR2(1000);
 DOC DBE_XMLDOM.DOMDOCUMENT;
 ELE DBE_XMLDOM.DOMELEMENT;
```

```

BEGIN
 FPATH := 'dir/simplexml.xml';
 SRC := '<ROOT>
 <A ATTR1="A_VALUE">
 <ACHILD>ACHILD TXT</ACHILD>

 B TXT
 </C/>
</ROOT>';
 DOC := DBE_XMLDOM.NEWDOMDOCUMENT(SRC);
 ELE := DBE_XMLDOM.GETDOCUMENTELEMENT(DOC);
 DBE_XMLDOM.WRITETOFILE(DBE_XMLDOM.MAKENODE(ELE), FPATH, 'ASCII');
 DBE_XMLDOM.FREEDOCUMENT(DOC);
END;
/
drop directory dir;

```

- **DBE\_XMLDOM.GETSESSIONTREENUM**

Queries the number of DOM trees of all types in the current session. The prototype of the DBE\_XMLDOM.GETSESSIONTREENUM function is as follows:

```

DBE_XMLDOM.GETSESSIONTREENUM()
RETURN INTEGER;

```

 **NOTE**

For DOM trees that have used FREEElement and FREENode, this function still counts them.

**Example:**

-- Create three documents and obtain the number of DOM trees in the current session.

```

DECLARE
 doc DBE_XMLDOM.DOMDocument;
 doc2 DBE_XMLDOM.DOMDocument;
 doc3 DBE_XMLDOM.DOMDocument;

 buffer varchar2(1010);
BEGIN
 -- Create three documents.
 doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<root>
 <elem1 attr="attrtest">
 <elem2>Im text</elem2>
 <elem3>Im text too</elem3>
 </elem1>
 <elem4>Text</elem4>
</root>
');
 doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<computer size="ITX" price="19999">
 <cpu>Ryzen 9 3950X</cpu>
 <ram>32GBx2 DDR4 3200MHz</ram>
 <motherboard>ROG X570i</motherboard>
 <gpu>RTX2070 Super</gpu>
 <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
 <hdd>12TB WD Digital</hdd>
 <psu>CORSAIR SF750</psu>
 <case>LIANLI TU150</case>
</computer>
');
 doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<bookstore>
 <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
 <title>The Autobiography of Benjamin Franklin</title>
 <author>
 <first-name>Benjamin</first-name>
 <last-name>Franklin</last-name>
 </author>
 <price>8.99</price>
 </book>

```

```

<book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
 <title>The Confidence Man</title>
 <author>
 <first-name>Herman</first-name>
 <last-name>Melville</last-name>
 </author>
 <price>11.99</price>
</book>
<book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
 <title>The Gorgias</title>
 <author>
 <name>Plato</name>
 </author>
 <price>9.99</price>
</book>
</bookstore>
');

-- Print IDs.
DBE_OUTPUT.PRINT_LINE(doc.id);
DBE_OUTPUT.PRINT_LINE(doc2.id);
DBE_OUTPUT.PRINT_LINE(doc3.id);
-- Call functions and print them.
DBE_OUTPUT.PRINT_LINE(DBE_XMLDOM.GETSESSIONTREENUM());
-- Release the documents.
DBE_XMLDOM.FREEDOCUMENT(doc);
DBE_XMLDOM.FREEDOCUMENT(doc2);
DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/

```

- **DBE\_XMLDOM.GETDOCTREESINFO**

Queries the DOM tree information of the document type in the current session, such as the memory usage. The prototype of the DBE\_XMLDOM.GETDOCTREESINFO function is as follows:

```

DBE_XMLDOM.GETDOCTREESINFO()
RETURN VARCHAR2;

```

#### NOTE

This function collects statistics only on DOM tree nodes of the document type.

#### Example:

```

-- Create three documents and obtain the information about the document tree in the current session.
DECLARE
 doc DBE_XMLDOM.DOMDocument;
 doc2 DBE_XMLDOM.DOMDocument;
 doc3 DBE_XMLDOM.DOMDocument;

 buffer varchar2(1010);
BEGIN
 -- Create three documents.
 doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<root>
 <elem1 attr="attrtest">
 <elem2>Im text</elem2>
 <elem3>Im text too</elem3>
 </elem1>
 <elem4>Text</elem4>
</root>
');
 doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<computer size="ITX" price="19999">
 <cpu>Ryzen 9 3950X</cpu>
 <ram>32GBx2 DDR4 3200MHz</ram>
 <motherboard>ROG X570i</motherboard>
 <gpu>RTX2070 Super</gpu>
 <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
 <hdd>12TB WD Digital</hdd>

```

```

<psu>CORSAIR SF750</psu>
<case>LIANLI TU150</case>
</computer>
');
doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<bookstore>
 <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
 <title>The Autobiography of Benjamin Franklin</title>
 <author>
 <first-name>Benjamin</first-name>
 <last-name>Franklin</last-name>
 </author>
 <price>8.99</price>
 </book>
 <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
 <title>The Confidence Man</title>
 <author>
 <first-name>Herman</first-name>
 <last-name>Melville</last-name>
 </author>
 <price>11.99</price>
 </book>
 <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
 <title>The Gorgias</title>
 <author>
 <name>Plato</name>
 </author>
 <price>9.99</price>
 </book>
</bookstore>
');

-- Print IDs.
DBE_OUTPUT.PRINT_LINE(doc.id);
DBE_OUTPUT.PRINT_LINE(doc2.id);
DBE_OUTPUT.PRINT_LINE(doc3.id);
-- Call functions and print them.
DBE_OUTPUT.PRINT_LINE(DBE_XMLDOM.GETDOCTREESINFO());
-- Release the documents.
DBE_XMLDOM.FREEDOCUMENT(doc);
DBE_XMLDOM.FREEDOCUMENT(doc2);
DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/

```

- DBE\_XMLDOM.GETDETAILDOCTREEINFO

Queries the number of subnodes of each type in the transferred document. The prototype of the DBE\_XMLDOM.GETDETAILDOCTREEINFO function is as follows:

```

DBE_XMLDOM.GETDETAILDOCTREEINFO(
 doc IN DOMDOCUMENT
)
RETURN VARCHAR2;

```

**Table 10-369** DBE\_XMLDOM.GETDETAILDOCTREEINFO parameters

Parameter	Description
doc	Specified DOMDocument node

 **NOTE**

This function collects statistics only on DOM tree nodes of the document type.

**Example:**

```
-- Create three documents and use this function to obtain the number of nodes of each type in each document.
DECLARE
doc DBE_XMLDOM.DOMDocument;
doc2 DBE_XMLDOM.DOMDocument;
doc3 DBE_XMLDOM.DOMDocument;

buffer varchar2(1010);
BEGIN
-- Create three documents.
doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<root>
 <elem1 attr="attrtest">
 <elem2>Im text</elem2>
 <elem3>Im text too</elem3>
 </elem1>
 <elem4>Text</elem4>
</root>
');
doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<computer size="ITX" price="19999">
 <cpu>Ryzen 9 3950X</cpu>
 <ram>32GBx2 DDR4 3200MHz</ram>
 <motherboard>ROG X570i</motherboard>
 <gpu>RTX2070 Super</gpu>
 <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
 <hdd>12TB WD Digital</hdd>
 <psu>CORSAIR SF750</psu>
 <case>LIANLI TU150</case>
</computer>
');
doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<bookstore>
 <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
 <title>The Autobiography of Benjamin Franklin</title>
 <author>
 <first-name>Benjamin</first-name>
 <last-name>Franklin</last-name>
 </author>
 <price>8.99</price>
 </book>
 <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
 <title>The Confidence Man</title>
 <author>
 <first-name>Herman</first-name>
 <last-name>Melville</last-name>
 </author>
 <price>11.99</price>
 </book>
 <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
 <title>The Gorgias</title>
 <author>
 <name>Plato</name>
 </author>
 <price>9.99</price>
 </book>
</bookstore>
');

-- Print IDs.
DBE_OUTPUT.PRINT_LINE(doc.id);
DBE_OUTPUT.PRINT_LINE(doc2.id);
DBE_OUTPUT.PRINT_LINE(doc3.id);
-- Call functions and print them.
buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc);
DBE_OUTPUT.PRINT_LINE(buffer);
buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc2);
DBE_OUTPUT.PRINT_LINE(buffer);
```

```

buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc3);
DBE_OUTPUT.PRINT_LINE(buffer);
-- Release the documents.
DBE_XMLDOM.FREEDOCUMENT(doc);
DBE_XMLDOM.FREEDOCUMENT(doc2);
DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/

```

### 10.11.2.14 DBE\_XMLPARSER

#### API Description

The DBE\_XMLPARSER API is used to deserialize XML character strings and convert the character strings that store XML documents to document nodes. [Table 10-370](#) lists all APIs supported by the DBE\_XMLPARSER advanced package.

The XMLParser data type can be used to store XMLParser data. The maximum number of XMLParser data records that can be stored is 16777215. The XMLPARSER data type can parse and create the DOMDocument node according to the input character string. The advanced package also provides the corresponding set and get APIs to perform operations on the constraint attributes of the parsing process.

#### NOTE

When the DBE\_XMLPARSER advanced package is used in the database whose character set is set to **SQL\_ASCII**, an error is reported if characters beyond the ASCII range are input.

The DBE\_XMLPARSER advanced package supports only the O-compatible mode.

**Table 10-370** DBE\_XMLPARSER parameters

API	Description
<a href="#">DBE_XMLPARSER.FREEPARSER</a>	Frees a parser.
<a href="#">DBE_XMLPARSER.GETDOCUMENT</a>	Obtains the parsed document node.
<a href="#">DBE_XMLPARSER.GETVALIDATIONMODE</a>	Obtains the validation attribute.
<a href="#">DBE_XMLPARSER.NEWPARSER</a>	Creates a parser instance.
<a href="#">DBE_XMLPARSER.PARSEBUFFER</a>	Parses the VARCHAR string.
<a href="#">DBE_XMLPARSER.PARSECLOB</a>	Parses the CLOB string.
<a href="#">DBE_XMLPARSER.SETVALIDATIONMODE</a>	Sets the validation attribute.

- [DBE\\_XMLPARSER.FREEPARSER](#)  
Frees a given parser object.  
The stored procedure prototype of [DBE\\_XMLPARSER.FREEPARSER](#) is as follows:

```
DBE_XMLPARSER.FREEPARSER (
 p IN parser);
```

**Table 10-371** DBE\_XMLPARSER.FREEPARSER parameters

Parameter	Description
p	Parser object

**Example:**

```
-- Create a parser and then release it.
DECLARE
 l_parser dbe_xmlparser.parser;
BEGIN
 l_parser := dbe_xmlparser.newparser();
-- Directly release the l_parser instance.
 dbe_xmlparser.freeparser(l_parser);
END;
/
```

Result: The operation is successful.

- **DBE\_XMLPARSER.GETDOCUMENT**

Returns the root node of the DOM tree document constructed by the parser. This function can be called only after the document is parsed.

The prototype of the DBE\_XMLPARSER.GETDOCUMENT function is as follows:

```
DBE_XMLPARSER.GETDOCUMENT (
 p IN parser)
RETURN DOMDocument;
```

**Table 10-372** DBE\_XMLPARSER.GETDOCUMENT parameters

Parameter	Description
p	Parser object

 **NOTE**

- If the GETDOCUMENT function has no input parameter, an error is reported.
- If the **parser** parameter of the GETDOCUMENT function is null, **NULL** is returned.
- If the parser input by the GETDOCUMENT function has not parsed any document, **NULL** is returned.

**Example:**

```
-- Create a parser to parse character strings and print the obtained document.
DECLARE
 l_parser dbe_xmlparser.parser;
 l_doc dbe_xmldom.domdocument;
 buffer varchar2 :=
'<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>';
 buffer2 varchar2;
```

```

BEGIN
L_parser := dbe_xmlparser.newparser();
dbe_xmlparser.PARSEBUFFER(L_parser, buffer);
L_doc := dbe_xmlparser.getdocument(L_parser);
-- The L_parser parses the character string and obtains the DOMDocument node through
GETDOCUMENT.
dbe_xmldom.writetobuffer(L_doc, buffer2);
RAISE NOTICE '%', buffer2;
-- Print the content in L_doc.
dbe_xmlparser.freeparser(L_parser);
dbe_xmldom.freedocument(L_doc);
END;
/

```

**Execution result:**

```

NOTICE: <?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>

```

- **DBE\_XMLPARSER.GETVALIDATIONMODE**

Obtains the parsing validation mode of a specified parser. If DTD validation is enabled, **TRUE** is returned. Otherwise, **FALSE** is returned.

The prototype of the DBE\_XMLPARSER.GETVALIDATIONMODE function is as follows:

```

DBE_XMLPARSER.GETVALIDATIONMODE (
p IN parser)
RETURN BOOLEAN;

```

**Table 10-373** DBE\_XMLPARSER.GETVALIDATIONMODE parameters

Parameter	Description
p	Parser object

**Example:**

```

-- Create a parser and use GETVALIDATIONMODE to check whether the parser validation mode is
enabled.
DECLARE
L_parser dbe_xmlparser.parser;
BEGIN
L_parser := dbe_xmlparser.newparser();
if (dbe_xmlparser.GETVALIDATIONMODE(L_parser) = true) then
RAISE NOTICE 'validation';
else
RAISE NOTICE 'no validation';
end if;
dbe_xmlparser.freeparser(L_parser);
END;
/

```

**Execution result:**

```

NOTICE: validation

```

- **DBE\_XMLPARSER.NEWPARSER**

Creates a parser object and returns a new parser instance.

The prototype of the DBE\_XMLPARSER.NEWPARSER function is as follows:

```
DBE_XMLPARSER.NEWPARSER
RETURN Parser;
```

**Example:**

```
-- Create a parser to parse character strings and then free the parser.
DECLARE
-- Create a parser.
l_parser dbe_xmlparser.parser;
l_doc dbe_xmldom.domdocument;
buffer varchar2(1000) :=
'<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>';
buffer2 varchar2(1000);
BEGIN
l_parser := dbe_xmlparser.newparser();
-- Parse the document and create a new DOM document.
dbe_xmlparser.PARSEBUFFER(l_parser, buffer);

dbe_xmlparser.freeparser(l_parser);
END;
/
```

Result: The operation is successful.

- **DBE\_XMLPARSER.PARSEBUFFER**

Parses XML documents stored in strings.

The stored procedure prototype of DBE\_XMLPARSER.PARSEBUFFER is as follows:

```
DBE_XMLPARSER.PARSEBUFFER (
p IN parser,
doc IN VARCHAR2);
```

**Table 10-374** DBE\_XMLPARSER.PARSEBUFFER parameters

Parameter	Description
p	Parser object
doc	A string that stores XML documents

 NOTE

- The maximum length of a character string that can be parsed by the PARSEBUFFER function is 32767. If the length exceeds the maximum, an error is reported.
- Different from the ORA database, this database supports only UTF-8 in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsed, a warning appears but the execution is normal. For versions later than 1.9, an error is reported.
- DTD validation differences:
  - **!ATTLIST to type (CHECK|check|Check) "Ch..."** reports an error because the default value "Ch..." is not an enumerated value in the brackets. However, the ORA database does not report this error.
  - **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer** reports an error because the letters are case-sensitive. **Baidu** cannot correspond to **baidu**. However, the ORA database does not report this error.
- Namespace validation difference: Undeclared namespace tags are parsed. However, the ORA database reports an error.
- Difference in parsing XML predefined entities: **&apos;** and **&quot;**; are parsed and translated into ' and ". However, predefined entities in ORA database are not translated into characters.

## Example:

```
-- Create a parser to parse character strings and print the obtained document.
```

```
DECLARE
 l_parser dbexmlparser.parser;
 l_doc dbexmlDOM.domdocument;
 buffer varchar2 :=
'<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>';
 buffer2 varchar2;
BEGIN
 l_parser := dbexmlparser.newparser();
 dbexmlparser.PARSEBUFFER(l_parser, buffer);
 l_doc := dbexmlparser.getdocument(l_parser);

 dbexmlDOM.writetobuffer(l_doc, buffer2);
 RAISE NOTICE '%', buffer2;

 dbexmlparser.freeparser(l_parser);
 dbexmlDOM.freedocument(l_doc);
END;
/
```

## Execution result:

```
NOTICE: <?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>
```

- **DBE\_XMLPARSER.PARSECLOB**

Parses XML documents stored in a CLOB.

The stored procedure prototype of DBE\_XMLPARSER.PARSECLOB is as follows:

```
DBE_XMLPARSER.PARSECLOB (
 p IN parser,
 doc IN CLOB);
```

**Table 10-375** DBE\_XMLPARSER.PARSECLOB parameters

Parameter	Description
p	Parser object
doc	A CLOB that stores XML documents

 **NOTE**

- PARSECLOB cannot parse CLOBs larger than 1 GB.
- Different from the ORA database, this database supports only UTF-8 in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsed, a warning appears but the execution is normal. For versions later than 1.9, an error is reported.
- DTD validation differences:
  - **!ATTLIST to type (CHECK|check|Check) "Ch..."** reports an error because the default value "Ch..." is not an enumerated value in the brackets. However, the ORA database does not report this error.
  - **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer** reports an error because the letters are case-sensitive. **Baidu** cannot correspond to **baidu**. However, the ORA database does not report this error.
- Namespace validation difference: Undeclared namespace tags are parsed. However, the ORA database reports an error.
- Difference in parsing XML predefined entities: **&apos;** and **&quot;** are parsed and translated into ' and ". However, predefined entities in ORA database are not translated into characters.

**Example:**

```
-- Create a parser to parse character strings and print the obtained document.
DECLARE
L_clob clob :=
'<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>this weekend!</body>
</note>';
-- Create a parser.
L_parser dbe_xmlparser.parser;
L_doc dbe_xmldom.domdocument;
buffer varchar2(1000);
BEGIN
L_parser := dbe_xmlparser.newparser();
-- Parse the document and create a new DOM document.
dbe_xmlparser.parseclob(L_parser, L_clob);
L_doc := dbe_xmlparser.getdocument(L_parser);
dbe_xmldom.writetobuffer(L_doc, buffer);
RAISE NOTICE '%',buffer;

dbe_xmlparser.freeparser(L_parser);
dbe_xmldom.freedocument(L_doc);

END;
/
```

**Execution result:**

```
NOTICE: <?xml version="1.0" encoding="UTF-8"?>
<note>
```

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>this weekend!</body>
</note>
```

- DBE\_XMLPARSER.SETVALIDATIONMODE

Sets the parsing validation mode of a specified parser.

The stored procedure prototype of DBE\_XMLPARSER.SETVALIDATIONMODE is as follows:

```
DBE_XMLPARSER.SETVALIDATIONMODE(
 p IN parser)
yes IN BOOLEAN);
```

**Table 10-376** DBE\_XMLPARSER.SETVALIDATIONMODE parameters

Parameter	Description
p	Parser object
yes	Mode to be set: <ul style="list-style-type: none"> <li>• <b>TRUE</b>: DTD validation is enabled.</li> <li>• <b>FALSE</b>: DTD validation is disabled.</li> </ul>

 **NOTE**

- If the input parameter **yes** of the SETVALIDATIONMODE function is null, the parsing validation mode of the parser is not changed.
- By default, the DTD validation is enabled during parser initialization.

**Example 1:**

```
-- Create a parser. The XML character string to be parsed does not match the DTD format.
-- If setValidationMode is set to false, the string can be parsed. If setValidationMode is set to true,
an error is reported during parsing.
DECLARE
 l_clob clob :=
'<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<table>
<name attr1="WEB" attr2="web2">African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>';
 l_parser dbe_xmlparser.parser;
 l_doc dbe_xmldom.domdocument;
 buffer varchar2(1000);
BEGIN
 l_parser := dbe_xmlparser.newparser();
 -- Set it to false for parsing.
 dbe_xmlparser.setValidationMode(l_parser, false);
 dbe_xmlparser.parseclob(l_parser, l_clob);
 l_doc := dbe_xmlparser.getdocument(l_parser);
 dbe_xmldom.writetobuffer(l_doc, buffer);
 RAISE NOTICE '%', buffer;
 dbe_xmlparser.freeparser(l_parser);
```

```
 dbe_xmldom.freedocument(L_doc);
END;
/
```

**Execution result:**

```
NOTICE: <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
<!ELEMENT note (to , from , heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
>]
<table>
<name attr1="WEB" attr2="web2">African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>
```

**Example 2:**

```
-- Parse a parser. The XML character string to be parsed does not match the DTD format.
-- An error is reported during parsing after setValidationMode is set to true.
DECLARE
 L_clob clob :=
 '<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
>]
<table>
<name attr1="WEB" attr2="web2">African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>';
 L_parser dbe_xmlparser.parser;
 L_doc dbe_xmldom.domdocument;
 buffer varchar2(1000);
BEGIN
 L_parser := dbe_xmlparser.newparser();
 -- Set it to true for parsing.
 --The XML character string does not match the DTD format. An error is expected.
 dbe_xmlparser.setValidationMode(L_parser, true);
 dbe_xmlparser.parseclob(L_parser, L_clob);
 L_doc := dbe_xmlparser.getdocument(L_parser);
 dbe_xmldom.writetobuffer(L_doc, buffer);
 dbe_xmlparser.freeparser(L_parser);
 dbe_xmldom.freedocument(L_doc);
END;
/
```

**Execution result:**

```
An error is reported during xmlparser parsing.
ERROR: invalid XML document
```

## 10.12 Retry Management

Retry is a process in which the database executes a SQL statement or stored procedure (including anonymous block) again in the case of execution failure, improving the execution success rate and user experience. The database checks the error code and retry configuration to determine whether to retry.

- If the execution fails, the system rolls back the executed statements and executes the stored procedure again.

Example:

```
gaussdb=# CREATE OR REPLACE PROCEDURE retry_basic (IN x INT)
AS
BEGIN
 INSERT INTO t1 (a) VALUES (x);
 INSERT INTO t1 (a) VALUES (x+1);
END;
/
gaussdb=# CALL retry_basic(1);
```

## 10.13 Debugging

### Syntax

RAISE has the following five syntax formats:

Figure 10-34 raise\_format::=

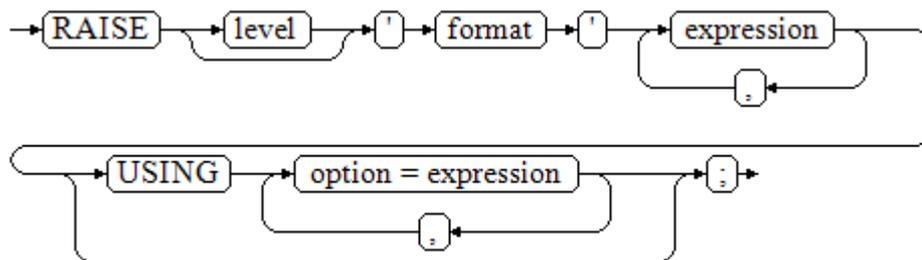


Figure 10-35 raise\_condition::=

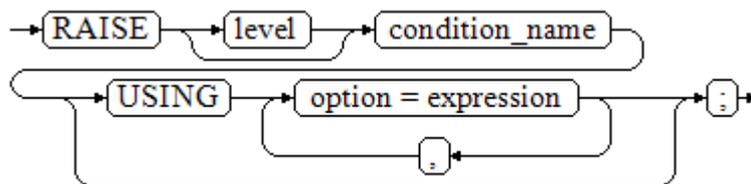


Figure 10-36 raise\_sqlstate::=

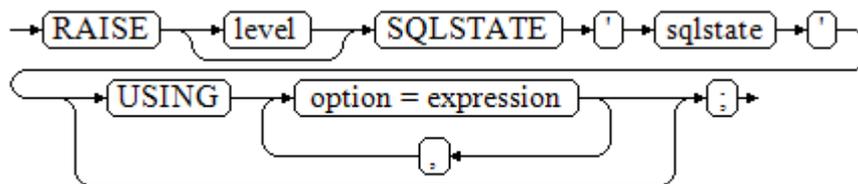


Figure 10-37 raise\_option::=

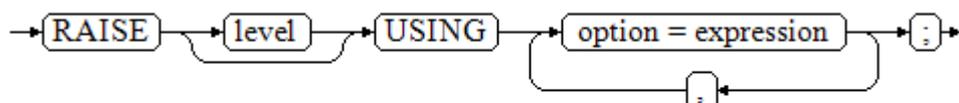
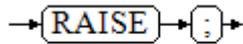


Figure 10-38 raise::=

**Parameter description:**

- The **level** option is used to specify the error level, that is, **DEBUG**, **LOG**, **INFO**, **NOTICE**, **WARNING**, or **EXCEPTION** (default). **EXCEPTION** throws an error that normally terminates the current transaction and the others only generate information at their levels. The GUC parameters **log\_min\_messages** and **client\_min\_messages** control whether the error messages of specific levels are reported to the client and are written to the server log.
- **format**: specifies the error message text to be reported, a format string. The format string can be appended with an expression for insertion to the message text. In a format string, **%** is replaced by the parameter value attached to format and **%%** is used to print **%**. For example:  
--v\_job\_id replaces % in the string.  

```
RAISE NOTICE 'Calling cs_create_job(%)',v_job_id;
```
- **option = expression**: inserts additional information to an error report. The **keyword** option can be **MESSAGE**, **DETAIL**, **HINT**, or **ERRCODE**, and each expression can be any string.
  - **MESSAGE**: specifies the error message text. This option cannot be used in a **RAISE** statement that contains a format character string in front of **USING**.
  - **DETAIL**: specifies detailed information of an error.
  - **HINT**: prints hint information.
  - **ERRCODE**: designates an error code (SQLSTATE) to a report. A condition name or a five-character SQLSTATE error code can be used.
- **condition\_name**: specifies the condition name corresponding to the error code.
- **sqlstate**: specifies the error code.

If neither a condition name nor an SQLSTATE is specified in the **RAISE EXCEPTION** command, the **RAISE EXCEPTION (P0001)** is used by default. If no message text is specified, the condition name or SQLSTATE is used as the message text by default.

**NOTICE**

- If the SQLSTATE specifies an error code, the error code is not limited to a defined error code. It can be any error code containing five digits or ASCII uppercase rather than **00000**. Do not use an error code ended with three zeros because such error codes are category codes and can be captured by the whole category.
- In O-compatible mode, SQLCODE is equivalent to SQLSTATE.

 NOTE

The syntax described in [Figure 10-38](#) does not append any parameter. This form is used only for the **EXCEPTION** statement in a **BEGIN** block so that the error can be re-processed.

## Examples

Display error and hint information when a transaction terminates:

```
CREATE OR REPLACE PROCEDURE proc_raise1(user_id in integer)
AS
BEGIN
RAISE EXCEPTION 'Noexistence ID --> %',user_id USING HINT = 'Please check your user ID';
END;
/

call proc_raise1(300011);

-- Execution result
ERROR: Noexistence ID --> 300011
HINT: Please check your user ID
```

Two methods are available for setting SQLSTATE:

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)
AS
BEGIN
RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';
END;
/

\set VERBOSITY verbose
call proc_raise2(300011);

-- Execution result
ERROR: Duplicate user ID: 300011
SQLSTATE: 23505
```

If the main parameter is a condition name or SQLSTATE, the following applies:

```
RAISE division_by_zero;
```

```
RAISE SQLSTATE '22012';
```

For example:

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)
AS
DECLARE
res int;
BEGIN
IF dividend=0 THEN
RAISE division_by_zero;
RETURN;
ELSE
res := div/dividend;
RAISE INFO 'division result: %', res;
RETURN;
END IF;
END;
/

call division(3,0);

-- Execution result
ERROR: division_by_zero
```

Alternatively:

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

# 11 Autonomous Transaction

An autonomous transaction is an independent transaction that is started during the execution of a primary transaction. Committing and rolling back an autonomous transaction does not affect the data that has been committed by the primary transaction. In addition, an autonomous transaction is not affected by the primary transaction.

Autonomous transactions are defined in stored procedures, functions, and anonymous blocks, and are declared using the **PRAGMA AUTONOMOUS\_TRANSACTION** keyword.

## 11.1 Stored Procedure Supporting Autonomous Transaction

An autonomous transaction can be defined in a stored procedure. The identifier of an autonomous transaction is **PRAGMA AUTONOMOUS\_TRANSACTION**. The syntax of an autonomous transaction is the same as that of creating a stored procedure. The following is an example.

```
-- Create a table.
gaussdb=# create table t2(a int, b int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# insert into t2 values(1,2);
INSERT 0 1
gaussdb=# select * from t2;
 a | b
---+---
 1 | 2
(1 row)

-- Create a stored procedure that contains an autonomous transaction.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_4(a int, b int) AS
DECLARE
 num3 int := a;
 num4 int := b;
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 insert into t2 values(num3, num4);
 db_output.print_line('just use call.');
```

```
-- Create a common stored procedure that calls an autonomous transaction stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_5(a int, b int) AS
DECLARE
BEGIN
 dbe_output.print_line('just no use call. ');
 insert into t2 values(666, 666);
 autonomous_4(a,b);
 rollback;
END;
/
CREATE PROCEDURE
-- Call a common stored procedure.
gaussdb=# select autonomous_5(11,22);
just no use call.
just use call.
autonomous_5

(1 row)
-- View the table result.
gaussdb=# select * from t2 order by a;
 a | b
----+----
 1 | 2
 11 | 22
(2 rows)
```

In the preceding example, a stored procedure containing an autonomous transaction is finally executed in a transaction block to be rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by the autonomous transaction.

## 11.2 Anonymous Block Supporting Autonomous Transaction

An autonomous transaction can be defined in an anonymous block. The identifier of an autonomous transaction is **PRAGMA AUTONOMOUS\_TRANSACTION**. The syntax of an autonomous transaction is the same as that of creating an anonymous block. The following is an example.

```
gaussdb=# create table t1(a int ,b text);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# START TRANSACTION;
START TRANSACTION
gaussdb=# DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 dbe_output.print_line('just use call. ');
 insert into t1 values(1,'you are so cute,will commit!');
END;
/
just use call.
ANONYMOUS BLOCK EXECUTE
gaussdb=# insert into t1 values(1,'you will rollback!');
INSERT 0 1
gaussdb=# rollback;
ROLLBACK

gaussdb=# select * from t1;
 a | b
```

```
-----+-----
1 | you are so cute,will commit!
(1 row)
```

In the preceding example, an anonymous block containing an autonomous transaction is finally executed before a transaction block to be rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by the autonomous transaction.

## 11.3 Function Supporting Autonomous Transaction

An autonomous transaction can be defined in a function. The identifier of an autonomous transaction is **PRAGMA AUTONOMOUS\_TRANSACTION**. The syntax of an autonomous transaction is the same as that of creating a function. The following is an example.

```
gaussdb=# create table t4(a int, b int, c text);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# CREATE OR REPLACE function autonomous_32(a int ,b int ,c text) RETURN int AS
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 insert into t4 values(a, b, c);
 return 1;
END;
/

gaussdb=# CREATE OR REPLACE function autonomous_33(num1 int) RETURN int AS
DECLARE
 num3 int := 220;
 tmp int;
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 num3 := num3/num1;
 return num3;
EXCEPTION
 WHEN division_by_zero THEN
 select autonomous_32(num3, num1, sqlerrm) into tmp;
 return 0;
END;
/
CREATE FUNCTION

gaussdb=# select autonomous_33(0);
autonomous_33
-----+-----
0
(1 row)

gaussdb=# select * from t4;
a | b | c
-----+-----
220 | 0 | division by zero
(1 row)
```

## 11.4 Restrictions

### CAUTION

- When an autonomous transaction is executed, an autonomous transaction session is started in the background. You can use **max\_concurrent\_autonomous\_transactions** to set the maximum number of concurrent autonomous transactions. The value range is 0 to 10000, and the default value is **10**.
- When **max\_concurrent\_autonomous\_transactions** is set to **0**, autonomous transactions cannot be executed.
- After a new session is started for an autonomous transaction, the default session parameters are used and objects (including session-level variables, local temporary variables, and global temporary table data) of the primary session are not shared.
- Theoretically, the upper limit of autonomous transactions is 10000. Actually, the upper limit is a dynamic value. For details, see the description of the GUC parameter **max\_concurrent\_autonomous\_transactions**.
- Autonomous transactions are affected by the communication buffer. The size of the information returned to the client is limited by the length of the communication buffer. If the size exceeds the length of the communication buffer, an error is reported.
- During the execution of an autonomous transaction, the primary transaction adds a lock to the autonomous transaction. If this lock remains due to an exception, the residual lock needs to be cleared by **gs\_clean**. The clearing period is controlled by the GUC parameter **gs\_clean\_timeout**, and the default value is 60 seconds. The lock of an autonomous transaction is not affected by lock timeout. The lock timeout interval is 2,147,483 seconds. If the execution time of an autonomous transaction exceeds this interval, an error is reported, indicating that the lock times out.
- For an autonomous transaction, the connection timeout interval is 5s and the number of connection attempts is 5. Signals are not responded to immediately during connection establishment. Signals are checked before each connection attempt. Timeout errors may occur in high concurrency, high CPU usage, high memory usage, and thread pool scale-out scenarios.

- A trigger function does not support autonomous transactions.

```
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 int, id2 int, id3 int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'id1' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
```

```
gaussdb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO test_trigger_des_tbl VALUES(new.id1, new.id2, new.id3);
RETURN new;END$$ LANGUAGE plpgsql;
ERROR: Triggers do not support autonomous transactions
DETAIL: N/A
```

```
gaussdb=# DROP TABLE test_trigger_des_tbl;
DROP TABLE
```

- Autonomous transactions cannot be called by non-top-layer anonymous blocks (but can only be called by top-layer autonomous transactions, including stored procedures, functions, and anonymous blocks).

```
gaussdb=# CREATE TABLE t1(a INT ,b TEXT);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
```

```
gaussdb=# DECLARE
--PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
DBE_OUTPUT.PRINT_LINE('JUST USE CALL.');
```

```
INSERT INTO t1 VALUES(1,'CAN YOU ROLLBACK!');
END;
INSERT INTO t1 VALUES(2,'I WILL ROLLBACK!');
ROLLBACK;
END;
/
JUST USE CALL.
ANONYMOUS BLOCK EXECUTE
```

```
gaussdb=# SELECT * FROM t1;
a | b
----+----
(0 rows)
```

```
gaussdb=# DROP TABLE t1;
DROP TABLE
```

- Autonomous transactions do not support **ref\_cursor** parameter transfer.

```
gaussdb=# create table sections(section_ID int);
gaussdb=# insert into sections values(1);
```

```
gaussdb=# CREATE OR REPLACE function proc_sys_ref()
return SYS_REFCURSOR
```

```
IS
declare
PRAGMA AUTONOMOUS_TRANSACTION;
C1 SYS_REFCURSOR;
BEGIN
OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
return C1;
END;
```

```
/
ERROR: Autonomous function do not support ref cursor as return types or out, inout arguments.
DETAIL: N/A
```

```
gaussdb=# CREATE OR REPLACE function proc_sys_ref(OUT C2 SYS_REFCURSOR, OUT a int)
return SYS_REFCURSOR
```

```
IS
declare
PRAGMA AUTONOMOUS_TRANSACTION;
C1 SYS_REFCURSOR;
BEGIN
OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
return C1;
END;
```

```
/
ERROR: Autonomous function do not support ref cursor as return types or out, inout arguments.
DETAIL: N/A
```

- Distributed autonomous transactions of the IMMUTABLE and STABLE types cannot be pushed down.  

```
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_test_in_p_116(num1 INT)
IMMUTABLE
AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
PERFORM pg_sleep(1);
END;
/
ERROR: Autonomous transactions do not support STABLE/IMMUTABLE.
DETAIL: Please remove stable/immutable.

gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_test_in_p_117(num1 INT) STABLE AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
PERFORM pg_sleep(1);
END;
/
ERROR: Autonomous transactions do not support STABLE/IMMUTABLE.
DETAIL: Please remove stable/immutable.
```
- The distributed system does not support detection. When a deadlock occurs, a lock waiting timeout error is reported.  

```
gaussdb=# CREATE TABLE test_lock (id INT,a DATE);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO test_lock VALUES (10,SYSDATE),(11,SYSDATE),(12,SYSDATE);
INSERT 0 3
gaussdb=# CREATE OR REPLACE FUNCTION autonomous_test_lock(num1 INT,num2 INT) RETURNS
INTEGER LANGUAGE plpgsql AS $$
DECLARE num3 INT := 4;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
UPDATE test_lock SET a=SYSDATE WHERE id =11;
RETURN num1+num2+num3;
END;
$$;
CREATE FUNCTION

gaussdb=# START TRANSACTION;
START TRANSACTION
gaussdb=# UPDATE test_lock SET a=SYSDATE WHERE id =11;
UPDATE 1
gaussdb=# CALL autonomous_test_lock(1,1);
ERROR: ERROR: Lock wait timeout: thread 139874535470848 on node datanode1 waiting for
ShareLock on transaction 16214 after 120000.124 ms
DETAIL: blocked by hold lock thread 139874577413888, statement <UPDATE test_lock SET a =
"sysdate"() WHERE id =11;>, hold lockmode ExclusiveLock.
CONTEXT: SQL statement "UPDATE test_lock SET a=SYSDATE WHERE id =11"
PL/SQL function autonomous_test_lock(integer,integer) line 5 at SQL statement
referenced column: autonomous_test_lock

gaussdb=# END;
ROLLBACK
gaussdb=# DROP TABLE test_lock;
DROP TABLE
```
- The autonomous transaction function cannot directly return the record type or the **out** output parameter and the record type at the same time.  

```
gaussdb=# CREATE OR REPLACE FUNCTION auto_func() RETURN RECORD
AS
DECLARE
TYPE rec_type IS RECORD(c1 INT, c2 INT);
r rec_type;
PRAGMA AUTONOMOUS_TRANSACTION;
```

```
BEGIN
 r.c1:=101;
 r.c2:=201;
 RETURN r;
END;
/
CREATE FUNCTION

gaussdb=# SELECT auto_func();
ERROR: unrecognized return type for PLSQL function.
CONTEXT: referenced column: auto_func
```

- The isolation level of an autonomous transaction cannot be changed.

```
gaussdb=# CREATE OR REPLACE PROCEDURE auto_func(r INT)
AS
DECLARE
 a INT;
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
 a:=r;
END;
/
CREATE FUNCTION
gaussdb=# call auto_func(1);
ERROR: ERROR: SET TRANSACTION ISOLATION LEVEL must be called before any query
CONTEXT: SQL statement "SET TRANSACTION ISOLATION LEVEL SERIALIZABLE"
PL/SQL function auto_func(integer) line 6 at SQL statement
referenced column: auto_func
```

- Autonomous transactions do not support the setof return type.

```
gaussdb=# CREATE OR REPLACE FUNCTION test_set() RETURN SETOF INT
AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 RETURN QUERY (SELECT unnest(ARRAY[ARRAY[1, 2], ARRAY[3, 4]]));
END;
/
ERROR: Autonomous transactions do not support RETURN SETOF.
DETAIL: N/A
```

# 12 System Catalogs and System Views

---

## 12.1 Overview of System Catalogs and System Views

System catalogs store structured metadata of GaussDB. They are the source of information used by GaussDB to control system running and are a core component of the database system.

System views provide ways to query the system catalogs and internal database status.

System catalogs and views are visible to either system administrators or all users. Some system catalogs and views have marked the need of administrator permissions, so they are accessible only to administrators.

You can delete and re-create system catalogs, add columns to them, and insert and update values in them, but doing so may make system information inconsistent and cause system faults. Generally, users should not modify system catalogs or system views, or rename their schemas. They are automatically maintained by the system.

---

### NOTICE

- You are advised not to modify the permissions on system catalogs or system views.
  - Do not add, delete, or modify system catalogs because doing so will result in exceptions or even cluster unavailability.
  - System catalogs and views do not support foreign key-related columns.
  - For details about column types in system catalogs and system views, see [Data Type](#).
- 

## 12.2 System Catalogs

## 12.2.1 GS\_AUDITING\_POLICY

**GS\_AUDITING\_POLICY** records the main information about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

**Table 12-1** GS\_AUDITING\_POLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
polname	name	Policy name, which must be unique
polcomments	name	Policy description field, which records policy-related description information and is represented by the <b>COMMENTS</b> keyword
modifydate	timestamp without time zone	The latest timestamp when a policy is created or modified
polenabled	boolean	Specifies whether to enable the policy. <ul style="list-style-type: none"><li>• <b>t</b> (true): enabled.</li><li>• <b>f</b> (false): disabled.</li></ul>

## 12.2.2 GS\_AUDITING\_POLICY\_ACCESS

**GS\_AUDITING\_POLICY\_ACCESS** records the DML database operations about the unified audit. Only the system administrator or security policy administrator can access this system catalog.

**Table 12-2** GS\_AUDITING\_POLICY\_ACCESS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
accesstype	name	DML database operation type, for example, <b>SELECT</b> , <b>INSERT</b> , and <b>DELETE</b> .
labelname	name	Specifies the resource label name, which corresponds to the <b>polname</b> column in the <a href="#">GS_AUDITING_POLICY</a> system catalog.

Name	Type	Description
policyoid	oid	OID of the audit policy, which corresponds to the OID in the <a href="#">12.2.1 GS_AUDITING_POLICY</a> system catalog.
modifydate	timestamp without time zone	Latest creation or modification timestamp.

### 12.2.3 GS\_AUDITING\_POLICY\_FILTERS

**GS\_AUDITING\_POLICY\_FILTERS** records the filtering policies about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

**Table 12-3** GS\_AUDITING\_POLICY\_FILTERS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
filtertype	name	Filter type. Currently, the value is <b>logical_expr</b> .
labelname	name	Name. Currently, the value is <b>logical_expr</b> .
policyoid	oid	OID of the audit policy, corresponding to the OID in the system catalog in <a href="#">GS_AUDITING_POLICY</a> .
modifydate	timestamp without time zone	Latest creation or modification timestamp.
logicaloperator	text	Logical character string of a filter criterion.

### 12.2.4 GS\_AUDITING\_POLICY\_PRIVILEGES

**GS\_AUDITING\_POLICY\_PRIVILEGES** records the DDL database operations about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

**Table 12-4** GS\_AUDITING\_POLICY\_PRIV columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
privilege_type	name	DDL database operation type, for example, <b>CREATE</b> , <b>ALTER</b> , and <b>DROP</b> .
labelname	name	Resource label name, which corresponds to the <b>polname</b> column in the <a href="#">12.2.1 GS_AUDITING_POLICY</a> system catalog.
policyoid	oid	Corresponds to OIDs in the <a href="#">12.2.1 GS_AUDITING_POLICY</a> system catalog.
modifydate	timestamp without time zone	Latest creation or modification timestamp.

## 12.2.5 GS\_ASP

GS\_ASP displays the persistent ACTIVE SESSION PROFILE samples. This system catalog can be queried only in the system library.

**Table 12-5** GS\_ASP columns

Name	Type	Description
sampleid	bigint	Sample ID.
sample_time	timestamp with time zone	Sampling time.
need_flush_sample	Boolean	Specifies whether the sample needs to be flushed to disks. <ul style="list-style-type: none"><li>● <b>t (true)</b>: yes.</li><li>● <b>f (false)</b>: no.</li></ul>
databaseid	oid	Database ID.
thread_id	bigint	Thread ID.
sessionid	bigint	Session ID.
start_time	timestamp with time zone	Start time of a session.
event	text	Specified event name.

Name	Type	Description
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.
tlevel	integer	Level of the streaming thread. The value corresponds to the level (ID) of the execution plan.
smpid	integer	Concurrent thread ID in SMP execution mode.
userid	oid	ID of a session user.
application_name	text	Name of the application.
client_addr	inet	IP address of a client.
client_hostname	text	Name of a client.
client_port	integer	TCP port number used by a client to communicate with the backend.
query_id	bigint	Debug query ID.
unique_query_id	bigint	Unique query ID.
user_id	oid	User ID in the key of the unique query.
cn_id	integer	A CN ID indicates a CN from which the unique SQL statement is obtained. The <b>cn_id</b> is in the key of the unique query.
unique_query	text	Standardized unique SQL text string
locktag	text	Information of a lock that the session waits for. It can be parsed using <b>locktag_decode</b> .
lockmode	text	Mode of a lock that the session waits for. <ul style="list-style-type: none"><li>● <b>LW_EXCLUSIVE</b>: exclusive lock.</li><li>● <b>LW_SHARED</b>: shared lock.</li><li>● <b>LW_WAIT_UNTIL_FREE</b>: waits for the <b>LW_EXCLUSIVE</b> to be available.</li></ul>

Name	Type	Description
block_sessionid	bigint	Blocks a session from obtaining the session ID of a lock if the session is waiting for the lock.
wait_status	text	Provides more details about the event column.
global_sessionid	text	Global session ID.
xact_start_time	timestamp with time zone	Start time of the transaction.
query_start_time	timestamp with time zone	Time when the statement starts to be executed.
state	text	Current statement state. The options are as follows: <ul style="list-style-type: none"> <li>• <b>active</b>: The backend is executing a query.</li> <li>• <b>idle in transaction</b>: The backend is in a transaction. However, no statement is being executed in the transaction.</li> <li>• <b>idle in transaction (aborted)</b>: The backend is in a transaction, but there are statements failed in the transaction.</li> <li>• <b>fastpath function call</b>: The backend is executing a fast-path function.</li> <li>• <b>disabled</b>: This state is reported if <b>track_activities</b> is disabled in this backend.</li> </ul>

## 12.2.6 GS\_CLIENT\_GLOBAL\_KEYS

**GS\_CLIENT\_GLOBAL\_KEYS** records information about the CMK in the encrypted equality feature. Each record corresponds to a CMK.

**Table 12-6** GS\_CLIENT\_GLOBAL\_KEYS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
global_key_name	name	CMK name
key_namespace	oid	A namespace OID that contains this CMK

Name	Type	Description
key_owner	oid	CMK owner
key_acl	aclitem[]	Access permissions that this key should have on creation
create_date	timestamp without time zone	Time when a key is created

## 12.2.7 GS\_CLIENT\_GLOBAL\_KEYS\_ARGS

**GS\_CLIENT\_GLOBAL\_KEYS\_ARGS** records the metadata about the CMK in the encrypted equality feature. Each record corresponds to a key-value pair of the CMK.

**Table 12-7** GS\_CLIENT\_GLOBAL\_KEYS\_ARGS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
global_key_id	oid	CMK OID
function_name	name	The value is <b>encryption</b> .
key	name	CMK metadata name
value	bytea	Value of the CMK metadata name

## 12.2.8 GS\_COLUMN\_KEYS

**GS\_COLUMN\_KEYS** records information about the CEK in the encrypted equality feature. Each record corresponds to a CEK.

**Table 12-8** GS\_COLUMN\_KEYS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
column_key_name	name	CEK name
column_key_distributed_id	oid	ID obtained based on the hash value of the fully qualified domain name (FQDN) of the CEK
global_key_id	oid	A foreign key, which is the CMK OID.

Name	Type	Description
key_namespace	oid	A namespace OID that contains this CEK
key_owner	oid	CEK owner
create_date	timestamp without time zone	Time when a CEK is created
key_acl	aclitem[]	Access permissions that this CEK should have on creation

## 12.2.9 GS\_COLUMN\_KEYS\_ARGS

**GS\_COLUMN\_KEYS\_ARGS** records the metadata about the CMK in the encrypted equality feature. Each record corresponds to a key-value pair of the CMK.

**Table 12-9** GS\_COLUMN\_KEYS\_ARGS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
column_key_id	oid	CEK OID
function_name	name	The value is <b>encryption</b> .
key	name	CEK metadata name
value	bytea	Value of the CEK metadata name

## 12.2.10 GS\_DATABASE\_LINK

**GS\_DATABASE\_LINK** stores database link information, mainly recording detailed information about database links. Only sysadmin can read this system catalog. Currently, the database link function is not supported.

**Table 12-10** GS\_DATABASE\_LINK columns

Name	Type	Description
oid	oid	Unique ID of the current database link object (hidden attribute, which must be specified).
dlname	name	Name of the current database link.
downer	oid	ID of the owner of the current database link. If the owner is <b>public</b> , the value is <b>0</b> .

Name	Type	Description
dlfdw	oid	OID of the foreign-data wrapper of the current database link.
dlcreator	oid	ID of the creator of the current database link.
options	text[]	Current database link connection information in the format of "keyword=value".
useroptions	text[]	User information used by the current database link to connect to the remote end, in the format of "keyword=value".
dlacl	aclitem[]	Current database link access permission.

## 12.2.11 GS\_DB\_PRIVILEGE

**GS\_DB\_PRIVILEGE** records the granting of ANY permissions. Each record corresponds to a piece of authorization information.

**Table 12-11** GS\_DB\_PRIVILEGE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
roleid	oid	User ID
privilege_type	text	ANY permission of a user. For details about the value, see <a href="#">Table 7-166</a> .
admin_option	boolean	Whether the ANY permission recorded in the <b>privilege_type</b> column can be re-granted <ul style="list-style-type: none"> <li>• t: yes</li> <li>• f: no</li> </ul>

## 12.2.12 GS\_DEPENDENCIES

**GS\_DEPENDENCIES** records object dependency information and has a one-to-many relationship with [12.2.13 GS\\_DEPENDENCIES\\_OBJ](#).

**Table 12-12** GS\_DEPENDENCIES columns

Name	Type	Description
schemaname	name	Name of a namespace
refobjpos	integer	Position of the referenced object <ul style="list-style-type: none"> <li>• 1: type</li> <li>• 2: package header</li> <li>• 4: function header</li> <li>• 8: function body</li> <li>• 16: package body</li> <li>• 32: view</li> </ul>
refobjoid	oid	OID of the referenced object
objectname	text	Name of the dependent object

### 12.2.13 GS\_DEPENDENCIES\_OBJ

GS\_DEPENDENCIES\_OBJ records the detailed information about the referenced object.

**Table 12-13** GS\_DEPENDENCIES\_OBJ columns

Name	Type	Description
schemaname	name	Name of a namespace.
type	integer	Type of the referenced object. <ul style="list-style-type: none"> <li>• 1: unknown type</li> <li>• 2: variable</li> <li>• 3: type</li> <li>• 4: function</li> <li>• 5: view</li> <li>• 6: function header</li> </ul>
name	text	Name of the referenced object.
objnode	pg_node_tree	Detailed information about the referenced object.

### 12.2.14 GS\_ENCRYPTED\_COLUMNS

GS\_ENCRYPTED\_COLUMNS records information about encrypted columns in the encrypted equality feature. Each record corresponds to an encrypted column.

**Table 12-14** GS\_ENCRYPTED\_COLUMNS columns

Name	Type	Description
rel_id	oid	Table OID
column_name	name	Name of an encrypted column.
column_key_id	oid	A foreign key, which is the CEK OID
encryption_type	tinyint	Encryption type. The value can be <b>2(DETERMINISTIC)</b> or <b>1(RANDOMIZED)</b> .
data_type_original_oid	oid	ID of the original data type of the encrypted column. For details about the values, see the oid columns in the <a href="#">PG_TYPE</a> system catalog.
data_type_original_mod	integer	Modifiers of the original data type of the encrypted column. For details about the values, see the atttypmod columns in the <a href="#">PG_ATTRIBUTE</a> system catalog.
create_date	timestamp without time zone	Time when an encrypted column is created

## 12.2.15 GS\_ENCRYPTED\_PROC

**GS\_ENCRYPTED\_PROC** provides information such as the parameters of encrypted functions and stored procedure functions, original data type of return values, and encrypted columns.

**Table 12-15** GS\_ENCRYPTED\_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
func_id	oid	OID of the function, corresponding to the OID row identifier in the <a href="#">12.2.81 PG_PROC</a> system catalog.
prorettype_orig	integer	Original data type of the return value.
last_change	timestamp without time zone	Last modification time of the encrypted function

Name	Type	Description
proargcachedcol	oidvector	OID of the encrypted column corresponding to the <b>INPUT</b> parameter of the function, corresponding to the OID row identifier in the <a href="#">12.2.14 GS_ENCRYPTED_COLUMNS</a> system catalog.
proallargtypes_orig	oid[]	Original data type of all function parameters.

## 12.2.16 GS\_GLOBAL\_CONFIG

**GS\_GLOBAL\_CONFIG** records the parameter values specified by users during cluster initialization. In addition, it also stores weak passwords set by users. Initial database users can write, modify, and delete parameters in system catalogs by using **ALTER** and **DROP**. Only the initial user, system administrator, and security administrator can access this system catalog. Other users do not have such permission.

**Table 12-16** GS\_GLOBAL\_CONFIG columns

Name	Type	Description
name	name	Specifies the preset parameter name, weak password name, or parameter required by users during cluster initialization.
value	text	Specifies the preset parameter value, weak password value, or parameter value required by users during cluster initialization.

## 12.2.17 GS\_JOB\_ATTRIBUTE

**GS\_JOB\_ATTRIBUTE** records attributes of DBE\_SCHEDULER scheduled tasks, including basic attributes of scheduled tasks, scheduled task classes, certificates, authorization, programs, and schedules. Common users do not have the permission to access the newly installed database cluster.

**Table 12-17** GS\_JOB\_ATTRIBUTE columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
job_name	text	Names of scheduled tasks, scheduled task classes, certificates, programs, and schedules, and authorized user names.

Name	Type	Description
attribute_name	text	Attribute names of scheduled tasks, scheduled task classes, certificates, programs, and schedules, and authorized content.
attribute_value	text	Attribute values of scheduled tasks, scheduled task classes, certificates, programs, and schedules.

## 12.2.18 GS\_JOB\_ARGUMENT

GS\_JOB\_ARGUMENT provides the parameter attributes of DBE\_SCHEDULER scheduled tasks and programs.

**Table 12-18** GS\_JOB\_ARGUMENT columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
argument_position	integer	Location of a parameter of a scheduled task or program.
argument_type	name	Parameter type of a scheduled task or program.
job_name	text	Name of a scheduled task or program.
argument_name	text	Parameter name of a scheduled task or program. The scheduled task inherits the parameter name of the program. Therefore, this parameter is null.
argument_value	text	Parameter value of a scheduled task. (The program cannot bind a value.)
default_value	text	Default parameter value of a program.

## 12.2.19 GS\_MASKING\_POLICY

GS\_MASKING\_POLICY records the main information about dynamic data masking policies. Each record corresponds to a masking policy. Only the system administrator or security policy administrator can access this system catalog.

**Table 12-19** GS\_MASKING\_POLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
polname	name	Policy name, which must be unique.
polcomments	name	Policy description field, which records policy-related description information and is represented by the <b>COMMENTS</b> keyword
modifydate	timestamp without time zone	Latest timestamp when a policy is created or modified.
polenabled	boolean	Specifies whether to enable the policy. <ul style="list-style-type: none"><li>• <b>t</b> (true): enabled</li><li>• <b>f</b> (false): disabled</li></ul>

## 12.2.20 GS\_MASKING\_POLICY\_ACTIONS

**GS\_MASKING\_POLICY\_ACTIONS** records the masking actions of a masking policy in the dynamic data masking policies. One masking policy corresponds to one or more rows of records in the catalog. Only the system administrator or security policy administrator can access this system catalog.

**Table 12-20** GS\_MASKING\_POLICY\_ACTIONS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
actiontype	name	Name of a masking function used by a masking policy
actparams	name	Parameter information transferred to a masking function
actlabelname	name	Name of a masked label
policyoid	oid	OID of a masking policy to which a record belongs, corresponding to the OID in <a href="#">GS_MASKING_POLICY</a> .
actmodifydate	timestamp without time zone	Latest timestamp when a record is created or modified

## 12.2.21 GS\_MASKING\_POLICY\_FILTERS

**GS\_MASKING\_POLICY\_FILTERS** records the user filtering criteria corresponding to the dynamic data masking policies. The corresponding masking policy takes effect only when the user information meets the filter criteria. Only the system administrator or security policy administrator can access this system catalog.

**Table 12-21** GS\_MASKING\_POLICY\_FILTERS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
filtertype	name	Filter type. Currently, the value is <b>logical_expr</b> .
filterlabelname	name	Filtering range. Currently, the value is <b>logical_expr</b> .
policyoid	oid	OID of a masking policy to which a record belongs, corresponding to the OID in <a href="#">GS_MASKING_POLICY</a> .
modifydate	timestamp without time zone	Latest timestamp when a user filter criterion is created or modified.
logicaloperator	text	Polish notation of the filter criteria.

## 12.2.22 GS\_MATVIEW

**GS\_MATVIEW** provides information about each materialized view in the database.

**Table 12-22** GS\_MATVIEW columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
matviewid	oid	OID of a materialized view.
mapid	oid	OID of a map table associated with a materialized view. Each map table corresponds to one materialized view. If a full materialized view does not correspond to a map table, the value of this column is <b>0</b> .

Name	Type	Description
ivm	boolean	Type of a materialized view. The value <b>t</b> indicates an incremental materialized view, and the value <b>f</b> indicates a full materialized view.
needrefresh	boolean	Reserved column.
refresh_time	timestamp without time zone	Last time when a materialized view was refreshed. If the materialized view is not refreshed, the value is null. This column is maintained only for incremental materialized views on DNs. In other cases, the value is null.

### 12.2.23 GS\_MATVIEW\_DEPENDENCY

**GS\_MATVIEW\_DEPENDENCY** provides association information about each incremental materialized view, base table, and Mlog table in the database. The Mlog table corresponding to the base table does not exist in the full materialized view. Therefore, no record is written into the Mlog table.

**Table 12-23** GS\_MATVIEW\_DEPENDENCY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
matviewid	oid	OID of a materialized view.
relid	oid	OID of a base table of a materialized view.
mlogid	oid	OID of a Mlog table which is the log table of a materialized view. Each Mlog table corresponds to one base table.
mxmin	integer	Reserved column.

### 12.2.24 GS\_MODEL\_WAREHOUSE

**GS\_MODEL\_WAREHOUSE** stores AI engine training models, including the models and detailed description of the training process.

**Table 12-24** GS\_MODEL\_WAREHOUSE columns

Name	Data Type	Description
oid	oid	Hidden column
modelname	name	Unique constraint
modelowner	oid	OID of a model owner
createtime	timestamp without time zone	Time when a model is created
processedtuples	integer	Number of tuples involved in training
discardedtuples	integer	Number of unqualified tuples not involved in training
preprocesstime	real	Data preprocessing time
exectime	real	Training duration
iterations	integer	Iteration round
outputtype	oid	OID of the output data type
modeltype	text	AI operator type
query	text	Query statement executed to create a model
modeldata	bytea	Stored binary model information
weight	real[]	Currently, this column applies only to GD operator models.
hyperparametersnames	text[]	Involved hyperparameter name
hyperparametersvalues	text[]	Hyperparameter value
hyperparametersoids	oid[]	OID of the data type corresponding to a hyperparameter
coefnames	text[]	Model parameter
coefvalues	text[]	Value of a model parameter
coefoids	oid[]	OID of the data type corresponding to a model parameter

Name	Data Type	Description
trainingscoresname	text[]	Method used to measure model performance
trainingscoresvalue	real[]	Value used to measure model performance
modeldescribe	text[]	Model description

## 12.2.25 GS\_OPT\_MODEL

**GS\_OPT\_MODEL** is a data table used when the AI engine is enabled to predict the planned time. It records the configurations, training results, features, corresponding system functions, and training history of machine learning models.

### NOTE

In the distributed scenario, this system catalog is provided, but the AI capabilities are unavailable.

## 12.2.26 GS\_PLAN\_TRACE

**GS\_PLAN\_TRACE** is used to store plan traces. It records details about the plan generation process of DML statements. Only users with the sysadmin permission can use this system catalog. The plan trace feature is not supported in the distributed scenario. Therefore, no data is displayed in this view in the distributed scenario.

**Table 12-25** GS\_PLAN\_TRACE columns

Name	Type	Description
query_id	text	Unique ID of the current request.
query	text	SQL statement of the current request. The value of this field cannot exceed the value of <b>track_activity_query_size</b> .
unique_sql_id	bigint	Unique ID of the SQL statement of the current request.
plan	text	Query plan text corresponding to the current request SQL statement. The value of this field cannot exceed 10 KB.
plan_trace	text	Details about the query plan generation process corresponding to the SQL statement of the current request. The value of this field cannot exceed 300 MB.
owner	oid	OID of the user who initiates the current SQL request.

Name	Type	Description
modifydate	timestamp with time zone	Time when the current plan trace is updated (that is, time when the plan trace is created).

## 12.2.27 GS\_POLICY\_LABEL

GS\_POLICY\_LABEL records the resource label configuration information. One resource label corresponds to one or more records, and each record identifies the resource label to which a database resource belongs. Only a system administrator or security policy administrator can access this system catalog.

Fully Qualified Domain Name (FQDN) identifies an absolute path of a database resource.

**Table 12-26** GS\_POLICY\_LABEL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
labelname	name	Specifies the resource label name.
labeltype	name	Resource tag type. Currently, the value is <b>RESOURCE</b> .
fqdnnamespace	oid	OID of a namespace to which an identified database resource belongs.
fqdnid	oid	OID of an identified database resource. If the database resource is a column, this column is the OID of the catalog.
relcolumn	name	Column name. If the identified database resource is a column, this column indicates the column name. Otherwise, this column is empty.
fqdnstype	name	Type of the identified database resource, for example, schema, table, column, or view.

## 12.2.28 GS\_RECYCLEBIN

GS\_RECYCLEBIN describes details about objects in the recycle bin of the flashback feature.

**Table 12-27** GS\_RECYCLEBIN columns

Name	Type	Description
oid	oid	System column
rcybaseid	oid	Base table object ID, which references <b>gs_recyclebin.oid</b>
rcydbid	oid	OID of the database to which the current object belongs
rcyrelid	oid	OID of the current object
rcyname	name	Name of the object in the recycle bin. The format is BIN\$ <i>unique_id</i> \$ <i>oid</i> \$. <i>unique_id</i> indicates the unique identifier with a maximum of 16 characters, and <i>oid</i> indicates the OID.
rcyoriginname	name	Original object name
rcyoperation	char	Operation type. The options are as follows: <ul style="list-style-type: none"> <li>● <b>d</b>: drop</li> <li>● <b>t</b>: truncate</li> </ul>
rcytype	integer	Object type. The options are as follows: <ul style="list-style-type: none"> <li>● <b>0</b>: table</li> <li>● <b>1</b>: index</li> <li>● <b>2</b>: TOAST table</li> <li>● <b>3</b>: TOAST index</li> <li>● <b>4</b>: sequence, indicating the sequence object that is automatically associated with the serial, bigserial, smallserial, and largeserial types.</li> <li>● <b>5</b>: partition</li> <li>● <b>6</b>: global index.</li> <li>● <b>7</b>: materialized view.</li> </ul>
rcyrecyclecsn	bigint	CSN when an object is dropped or truncated
rcyrecycletime	timestamp with time zone	Time when an object is dropped or truncated
rcycreatecsn	bigint	CSN when an object is created
rcychangecsn	bigint	SCN when an object definition is modified
rcynamespace	oid	OID of the namespace that contains this relationship
rcyowner	oid	Owner of the relationship

Name	Type	Description
rcytablespace	oid	Tablespace in which this relationship is stored. If the value is <b>0</b> , the default tablespace of the database is used. This column is meaningless if the relationship has no on-disk file.
rcyrelfilenode	oid	File name of the recycle bin object on a disk, or <b>0</b> if none, which is used to restore the physical file when the TRUNCATE object is restored.
rcycanrestore	Boolean	Specifies whether flashback can be performed separately.
rcycanpurge	Boolean	Specifies whether the purge operation can be performed independently.
rcyfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table.
rcyfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table.

## 12.2.29 GS\_SQL\_PATCH

**GS\_SQL\_PATCH** records the status information about SQL\_PATCH on the current node.

**Table 12-28** GS\_SQL\_PATCH columns

Name	Type	Description
patch_name	name	Patch name.
unique_sql_id	bigint	Global unique query ID
owner	oid	ID of the user who creates the patch.
enable	boolean	Specifies whether the patch takes effect.
status	"char"	Patch status (reserved column).
abort	boolean	Specifies whether the patch is an abort hint.
hint_string	text	Hint text.
hint_node	pg_node_tree	Hint parsing and serialization result.
original_query	text	Original statement (reserved column).

Name	Type	Description
patched_query	text	Patched statement (reserved column).
original_query_tree	pg_node_tree	Original statement parsing result (reserved column).
patched_query_tree	pg_node_tree	Patched statement parsing result (reserved column).
description	text	Patch description.
parent_unique_sql_id	bigint	Globally unique ID of the outer statement of the SQL statement for which the patch takes effect. The value of this parameter is <b>0</b> for statements outside a stored procedure. For statements inside the stored procedure, and the value of this parameter is the globally unique ID of the statement that invokes the stored procedure.

### 12.2.30 GS\_TXN\_SNAPSHOT

**GS\_TXN\_SNAPSHOT** is a timestamp-CSN mapping table. It periodically samples and maintains an appropriate time range to estimate the CSN corresponding to the timestamp in the range.

**Table 12-29** GS\_TXN\_SNAPSHOT columns

Name	Type	Description
snptime	timestamp with time zone	Snapshot time
snpxmin	bigint	Minimum snapshot ID
snpcsn	bigint	Snapshot CSN
snpsnapshot	text	Serialized snapshot text

### 12.2.31 GS\_UID

**GS\_UID** records the unique identification meta information of the **hasuids** attribute table in the database.

**Table 12-30** GS\_UID columns

Name	Type	Description
relid	oid	OID of a table

Name	Type	Description
uid_backup	bigint	Largest unique identifier that can be assigned to a table

## 12.2.32 GS\_WORKLOAD\_RULE

The GS\_WORKLOAD\_RULE system catalog records information about SQL concurrency control rules. There is no permission restriction on this system catalog. All users can query this system catalog.

**Table 12-31** GS\_WORKLOAD\_RULE columns

Name	Type	Description
rule_id	bigint	Concurrency control rule ID, which is automatically generated by the system.
rule_name	name	Name of the concurrency control rule, which is used for search. The name may not be unique and can be <b>NULL</b> .
databases	name[]	List of databases on which the concurrency control rules take effect. If the value is <b>NULL</b> , the concurrency control rules take effect for all databases.
max_workload	bigint	Maximum number of concurrent rule settings.
is_valid	boolean	Determines whether the concurrency control rules take effect. If the concurrency control rules time out, the value is set to <b>false</b> .
start_time	timestamp with time zone	Start time of the concurrency control rules. The value <b>NULL</b> indicates that the rules take effect from now on.
end_time	timestamp with time zone	End time of the concurrency control rules. The value <b>NULL</b> indicates that the rules are always effective.
rule_type	text	Concurrency control rule type. Currently, only "sqlid", "select", "insert", "update", "delete", "merge", and "resource" are supported. Other values are invalid.
option_val	text[]	Parameter values of concurrency control rules, including SQL ID, keyword list, and resource restriction.  For details, see the description of the <a href="#">gs_add_workload_rule</a> API.

Name	Type	Description
node_names	text[]	List of nodes on which the concurrency control rules take effect. This parameter is reserved and does not take effect currently.
user_names	text[]	List of users for which the concurrency control rules take effect. This parameter is reserved and does not take effect currently.

## 12.2.33 PG\_AGGREGATE

PG\_AGGREGATE records information about aggregate functions. Each entry in PG\_AGGREGATE is an extension of an entry in PG\_PROC. The PG\_PROC entry carries the aggregate's name, input and output data types, and other information that is similar to ordinary functions.

**Table 12-32** PG\_AGGREGATE columns

Name	Type	Reference	Description
aggfnoid	regproc	<a href="#">PG_PROC.proname</a>	<a href="#">PG_PROC</a> proname of the aggregate function
aggtransfn	regproc	<a href="#">PG_PROC.proname</a>	Transition function
aggcollectfn	regproc	<a href="#">PG_PROC.proname</a>	Collect function
aggfinalfn	regproc	<a href="#">PG_PROC.proname</a>	Final function ( <b>0</b> if none)
aggstortop	oid	<a href="#">PG_OPERATOR.oid</a>	Associated sort operator ( <b>0</b> if none)
aggtranstype	oid	<a href="#">PG_TYPE.oid</a>	Data type of the aggregate function's internal transition (state) data  The possible values and their meanings are defined by the types in <a href="#">pg_type.h</a> . The main two types are polymorphic (isPolymorphicType) and non-polymorphic.
agginitval	text	-	Initial value of the transition state. This is a text column containing the initial value in its external string representation. If this column is null, the transition state value starts from null.

Name	Type	Reference	Description
agginitcollect	text	-	Initial value of the collection state. This is a text column containing the initial value in its external string representation. If this column is null, the collection state value starts from null.
aggkind	"char"	-	Type of the aggregate function: <ul style="list-style-type: none"> <li>• <b>n</b>: normal aggregate</li> <li>• <b>o</b>: ordered set aggregate</li> </ul>
aggnumdirect args	smallint	-	Number of direct parameters (non-aggregation-related parameters) of the aggregate function of the ordered set aggregate type. For an aggregate function of the normal aggregate type, the value is <b>0</b> .

## 12.2.34 PG\_AM

PG\_AM records information about index access methods. There is one row for each index access method supported by the system.

**Table 12-33** PG\_AM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
amname	name	-	Name of the access method.
amstrategies	smallint	-	Number of operator strategies for the access method ( <b>0</b> if the access method does not have a fixed set of operator strategies).
amsupport	smallint	-	Number of support routines for the access method.

Name	Type	Reference	Description
amcanorder	boolean	-	Specifies whether the access method supports ordered scans sorted by the indexed column's value. <ul style="list-style-type: none"> <li>• <b>t</b> (true): supported.</li> <li>• <b>f</b> (false): not supported.</li> </ul>
amcanorderbyop	boolean	-	Specifies whether the access method supports ordered scans sorted by the result of an operator on the indexed column. <b>t</b> (true): supported. <b>f</b> (false): not supported.
amcanbackward	boolean	-	Specifies whether the access method supports backward scanning. <ul style="list-style-type: none"> <li>• <b>t</b> (true): supported.</li> <li>• <b>f</b> (false): not supported.</li> </ul>
amcanunique	boolean	-	Specifies whether the access method supports unique indexes. <ul style="list-style-type: none"> <li>• <b>t</b> (true): supported.</li> <li>• <b>f</b> (false): not supported.</li> </ul>
amcanmulticol	boolean	-	Specifies whether the access method supports multi-column indexes. <ul style="list-style-type: none"> <li>• <b>t</b> (true): supported.</li> <li>• <b>f</b> (false): not supported.</li> </ul>
amoptionalkey	boolean	-	Specifies whether the access method supports scanning without any constraint for the first index column. <ul style="list-style-type: none"> <li>• <b>t</b> (true): supported.</li> <li>• <b>f</b> (false): not supported.</li> </ul>
amsearcharray	boolean	-	Specifies whether the access method supports <b>ScalarArrayOpExpr</b> searches. <b>t</b> (true): supported. <b>f</b> (false): not supported.

Name	Type	Reference	Description
amsearchnulls	boolean	-	Specifies whether the access method supports <b>IS NULL/NOT NULL</b> searches. <ul style="list-style-type: none"> <li>• <b>t</b> (true): supported.</li> <li>• <b>f</b> (false): not supported.</li> </ul>
amstorage	boolean	-	Specifies whether the index storage data type can differ from the column data type. <ul style="list-style-type: none"> <li>• <b>t</b> (true): same.</li> <li>• <b>f</b> (false): different.</li> </ul>
amclusterable	boolean	-	Specifies whether an index of this type can be clustered on. <ul style="list-style-type: none"> <li>• <b>t</b> (true): allowed.</li> <li>• <b>f</b> (false): not allowed.</li> </ul>
ampredlocks	boolean	-	Specifies whether an index of this type manages fine-grained predicate locks. <ul style="list-style-type: none"> <li>• <b>t</b> (true): allowed.</li> <li>• <b>f</b> (false): not allowed.</li> </ul>
amkeytype	oid	<b>oid</b> in <b>PG_TYPE</b>	Type of data stored in index ( <b>0</b> if it is not a fixed type)
aminsert	regproc	<b>prname</b> in <b>PG_PROC</b>	"Insert this tuple" function
ambeginscan	regproc	<b>prname</b> in <b>PG_PROC</b>	"Prepare for index scan" function
amgettupl	regproc	<b>prname</b> in <b>PG_PROC</b>	"Next valid tuple" function ( <b>0</b> if none)
amgetbitmap	regproc	<b>prname</b> in <b>PG_PROC</b>	"Fetch all valid tuples" function ( <b>0</b> if none)
amrescan	regproc	<b>prname</b> in <b>PG_PROC</b>	"(Re)start index scan" function
amendscan	regproc	<b>prname</b> in <b>PG_PROC</b>	"Clean up after index scan" function
ammarkpos	regproc	<b>prname</b> in <b>PG_PROC</b>	"Mark current scan position" function
amrestrpos	regproc	<b>prname</b> in <b>PG_PROC</b>	"Restore marked scan position" function
ammerge	regproc	<b>prname</b> in <b>PG_PROC</b>	"Merge multiple indexes" function

Name	Type	Reference	Description
ambuild	regproc	<b>prname</b> in <a href="#">PG_PROC</a>	"Build new index" function
ambuilde mpty	regproc	<b>prname</b> in <a href="#">PG_PROC</a>	"Build empty index" function
ambulkdel ete	regproc	<b>prname</b> in <a href="#">PG_PROC</a>	Bulk-delete function
amvacuum cleanup	regproc	<b>prname</b> in <a href="#">PG_PROC</a>	Post-VACUUM cleanup function
amcanretu rn	regproc	<b>prname</b> in <a href="#">PG_PROC</a>	Function to check whether the index supports index-only scans ( <b>0</b> if none)
amcostesti mate	regproc	<b>prname</b> in <a href="#">PG_PROC</a>	Function to estimate cost of an index scan
amoptions	regproc	<b>prname</b> in <a href="#">PG_PROC</a>	Function to parse and validate <b>reloptions</b> for an index

## 12.2.35 PG\_AMOP

PG\_AMOP records information about operators associated with access method operator families. There is one row for each operator that is a member of an operator family. A family member can be either a search operator or an ordering operator. An operator can appear in more than one family, but cannot appear in more than one search position nor more than one ordering position within a family.

**Table 12-34** PG\_AMOP columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
amopfamily	oid	<a href="#">PG_OPFAMILY</a> .oid	Operator family of this entry.
amoplefttype	oid	<b>oid</b> in <a href="#">PG_TYPE</a>	Left-hand input data type of the operator For details about the possible values and their descriptions, see <a href="#">7.3 Data Type</a> .

Name	Type	Reference	Description
amoprighthtype	oid	oid in <a href="#">PG_TYPE</a>	Right-hand input data type of the operator For details about the possible values and their descriptions, see <a href="#">7.3 Data Type</a> .
amopstrategy	smallint	-	Number of operator strategies.
amoppurpose	"char"	-	Purpose of the operator. <ul style="list-style-type: none"> <li>• s: search</li> <li>• o: order</li> </ul>
amopopr	oid	<a href="#">PG_OPERATOR</a> .oid	OID of the operator.
amopmethod	oid	<a href="#">PG_AM</a> .oid	Operator family of the index access method.
amopsortfamily	oid	<a href="#">PG_OPFAMILY</a> .oid	The B-tree operator family according to which this entry sorts for an ordering operator ( <b>0</b> for a search operator).

A search operator entry indicates that an index of this operator family can be searched to find all rows satisfying **WHERE indexed\_column operator constant**. Obviously, such an operator must return a Boolean value, and its left-hand input type must match the index's column data type.

An ordering operator entry indicates that an index of this operator family can be scanned to return rows in the order represented by **ORDER BY indexed\_column operator constant**. Such an operator could return any sortable data type, though again its left-hand input type must match the index's column data type. The exact semantics of **ORDER BY** are specified by the **amopsortfamily** column, which must reference the B-tree operator family for the operator's result type.

## 12.2.36 PG\_AMPROC

**PG\_AMPROC** records information about the support procedures associated with the access method operator families. There is one row for each support procedure that belongs to an operator family.

**Table 12-35** PG\_AMPROC columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)

Name	Type	Reference	Description
amprocfamily	oid	<a href="#">PG_OPFAMILY.oid</a>	Operator family of this entry
amproclefttype	oid	<a href="#">PG_TYPE.oid</a>	Left-hand input data type of the associated operator For details about common data types, see <a href="#">Data Type</a> .
amprocrighttype	oid	<a href="#">PG_TYPE.oid</a>	Right-hand input data type of the associated operator For details about common data types, see <a href="#">Data Type</a> .
amprocnum	smallint	-	Support procedure number
amproc	regproc	<a href="#">PG_PROC.proname</a>	OID of the procedure

The usual interpretation of the **amproclefttype** and **amprocrighttype** columns is that they identify the left and right input types of the operator(s) that a particular support procedure supports. For some access methods, these match the input data type(s) of the support procedure itself; for others not. There is a notion of "default" support procedures for an index, which are those with **amproclefttype** and **amprocrighttype** both equal to the index opclass's **opcintype**.

## 12.2.37 PG\_APP\_WORKLOADGROUP\_MAPPING

**PG\_APP\_WORKLOADGROUP\_MAPPING** provides load mapping group information in the database.

**Table 12-36** PG\_APP\_WORKLOADGROUP\_MAPPING columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
appname	name	Application name
workload_gpname	name	Mapped workload group name

## 12.2.38 PG\_ATTRDEF

**PG\_ATTRDEF** records default values of columns.

**Table 12-37** PG\_ATTRDEF columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
adrelid	oid	Table to which a column belongs
adnum	smallint	Number of columns
adbin	pg_node_tree	Internal representation of the default value of the column
adsrc	text	Internal representation of the human-readable default value
adgencol	"char"	Specifies whether a column is a generated column. The value <b>s</b> indicates that the column is a generated column, and the value '\0' indicates that the column is a common column. The default value is '\0'.

## 12.2.39 PG\_ATTRIBUTE

PG\_ATTRIBUTE records information about table columns.

**Table 12-38** PG\_ATTRIBUTE columns

Name	Type	Description
attrelid	oid	Table to which a column belongs.
attname	name	Column name.
atttypid	oid	Column type.
attstattarget	integer	Level of details of statistics collected for this column by <b>ANALYZE</b> . <ul style="list-style-type: none"><li>• The value <b>0</b> indicates that no statistics should be collected.</li><li>• A negative value indicates that the system default statistic object is used.</li><li>• The exact meaning of positive values is data type-dependent.</li></ul> For scalar data types, <b>attstattarget</b> is both the target number of "most common values" to collect, and the target number of histogram bins to create.
attlen	smallint	Copy of typlen in <a href="#">12.2.105 PG_TYPE</a> of the column type.

Name	Type	Description
attnum	smallint	Number of the column
attdims	integer	Number of dimensions if the column is an array ( <b>0</b> in other cases)
attcacheoff	integer	This column is always <b>-1</b> on disk. When it is loaded into a row descriptor in the memory, it may be updated to cache the offset of the columns in the row.
atttypmod	integer	Type-specific data supplied at the table creation time (for example, the maximum length of a <b>varchar</b> column). This column is used as the third parameter when passing to type-specific input functions and length coercion functions. The value will generally be <b>-1</b> for types that do not need ATTTYPMOD.
attbyval	boolean	Copy of typbyval in <a href="#">12.2.105 PG_TYPE</a> of the column type.
attstorage	"char"	Copy of typstorage in <a href="#">12.2.105 PG_TYPE</a> of the column type.
attalign	"char"	Copy of typalign in <a href="#">12.2.105 PG_TYPE</a> of the column type.
attnotnull	boolean	A NOT NULL constraint. It is possible to change this column to enable or disable the constraint.
atthasdef	boolean	This column has a default value, in which case there will be a corresponding entry in <a href="#">12.2.45 PG_ATTRDEF</a> that actually defines the value.
attisdropped	boolean	Indicates that this column has been deleted and is no longer valid. A deleted column is still physically present in the table but is ignored by the analyzer, so it cannot be accessed through SQL.
attislocal	boolean	Indicates that this column is locally defined in the relationship. Note that a column can be locally defined and inherited simultaneously.

Name	Type	Description
attcmpr mode	tinyint	Compressed modes for a specific column. The compressed mode includes: <ul style="list-style-type: none"> <li>• <b>0</b>: not compressed (<b>ATT_CMPR_NOCOMPRESS</b>)</li> <li>• <b>1</b>: DELTA compression algorithm (<b>ATT_CMPR_DELTA</b>)</li> <li>• <b>2</b>: dictionary compression algorithm (<b>ATT_CMPR_DICTIONARY</b>)</li> <li>• <b>3</b>: prefix compression algorithm (<b>ATT_CMPR_PREFIX</b>)</li> <li>• <b>4</b>: digital string compression algorithm (<b>ATT_CMPR_NUMSTR</b>)</li> </ul>
attinhcount	integer	Number of direct ancestors that this column has. A column with an ancestor cannot be dropped nor renamed.
attcollation	oid	Defined collation of a column.
attacl	aclitem[]	Permissions for column-level access.
attoptions	text[]	Column attribute. Currently, the following attributes are supported: <ul style="list-style-type: none"> <li>• <b>n_distinct</b>: number of <b>distinct</b> values of a column (excluding subtables).</li> <li>• <b>n_distinct_inherited</b>: number of <b>distinct</b> values of a column (including subtables).</li> </ul>
attfdwoptions	text[]	Column attribute of a foreign table. Currently, <b>dist_fdw</b> , <b>file_fdw</b> , and <b>log_fdw</b> do not use foreign table column attributes.
attinitdefval	bytea	<b>attinitdefval</b> stores the default value expression. <b>ADD COLUMN</b> in the row-store table must use this column.
attkvtype	tinyint	Specifies a key value type for a column. Types include: <ul style="list-style-type: none"> <li><b>0</b>: default value (<b>ATT_KV_UNDEFINED</b>)</li> <li><b>1</b>: dimension (<b>ATT_KV_TAG</b>)</li> <li><b>2</b>: indicator (<b>ATT_KV_FIELD</b>)</li> <li><b>3</b>: time column (<b>ATT_KV_TIMETAG</b>)</li> <li><b>4</b>: hidden distribution key (<b>ATT_KV_HIDETAG</b>)</li> </ul>

## 12.2.40 PG\_AUTHID

**PG\_AUTHID** records information about database authentication identifiers (roles). The concept of users is contained in that of roles. A user is actually a role whose

**rolcanlogin** has been set. Any role, whether its **rolcanlogin** is set or not, can use other roles as members.

For a cluster, only one **PG\_AUTHID** exists, which is not available for every database. This system catalog is accessible only to system administrators.

**Table 12-39** PG\_AUTHID columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
rolname	name	Name of a role
rolsuper	boolean	Whether the role is the initial system administrator with the highest permission <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolinherit	boolean	Whether the role automatically inherits permissions of roles of which it is a member <ul style="list-style-type: none"> <li>• <b>t</b> (true): automatically inherited</li> <li>• <b>f</b> (false): not automatically inherited</li> </ul>
rolcreatorole	boolean	Whether the role can create more roles <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolcreatedb	boolean	Whether the role can create databases <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolcatupdate	boolean	Whether the role can directly update system catalogs Only the initial system administrator whose <b>usesysid</b> is <b>10</b> has this permission. It is unavailable for other users. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolcanlogin	boolean	Whether the role can log in (whether this role can be given as the initial session authorization identifier) <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>

Name	Type	Description
rolreplication	boolean	Specifies whether the role has the replication permission. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolauditadmin	boolean	Specifies whether the role has the audit administrator permission. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolsystemadmin	boolean	Specifies whether the role has the system administrator permission. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolconnlimit	integer	Maximum number of concurrent connections that this role can make (valid for roles that can log in) The value <b>-1</b> indicates there is no limit.
rolpassword	text	Password (possibly encrypted) (null if no password)
rolvalidbegin	timestamp with time zone	Account validity start time ( <b>NULL</b> if no start time)
rolvaliduntil	timestamp with time zone	Password expiry time ( <b>NULL</b> if no expiration)
rolrespool	name	Resource pool that a user can use
roluseft	boolean	Whether the role can perform operations on foreign tables <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolparentid	oid	OID of a group user to which the user belongs
roltabspace	text	Maximum size of a user data table
rolkind	"char"	Type of a user. <ul style="list-style-type: none"> <li>• <b>p</b>: permanent user</li> <li>• <b>n</b>: common user</li> </ul>
roltemp space	text	Maximum size of a user's temporary table, in KB.
rolspill space	text	Maximum size of data that can be written to disks when a user executes a job, in KB.

Name	Type	Description
rolexpdata	text	Query rules that can be set by users (reserved)
rolmonitoradmin	boolean	Specifies whether the role has the monitor administrator permission. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
roloperatoradmin	boolean	Specifies whether the role has the O&M administrator permission. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
rolpolicyadmin	boolean	Specifies whether the role has the security policy administrator permission. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>

## 12.2.41 PG\_AUTH\_HISTORY

PG\_AUTH\_HISTORY records the authentication history of a role. This system catalog is accessible only to system administrators.

**Table 12-40** PG\_AUTH\_HISTORY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
roloid	oid	ID of a role.
passwordtime	timestamp with time zone	Time of password creation and change.
rolpassword	text	Ciphertext of the role password. The encryption mode is determined by the GUC parameter <b>password_encryption_type</b> .

## 12.2.42 PG\_AUTH\_MEMBERS

PG\_AUTH\_MEMBERS records the membership between roles.

**Table 12-41** PG\_AUTH\_MEMBERS columns

Name	Type	Description
roleid	oid	ID of a role that has a member.
member	oid	ID of a role that is a member of ROLEID.
grantor	oid	ID of a role that grants this membership.
admin_option	boolean	Specifies whether a member can grant membership in ROLEID to others. The value can be <b>true</b> (yes) and <b>false</b> (no).

## 12.2.43 PG\_CAST

PG\_CAST records the conversion relationship between data types.

**Table 12-42** PG\_CAST columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
castsource	oid	OID of the source data type
casttarget	oid	OID of the target data type
castfunc	oid	OID of the conversion function ( <b>0</b> if no conversion function is required)
castcontext	"char"	Conversion mode between the source and target data types. <ul style="list-style-type: none"> <li>• <b>e</b>: Only explicit conversion can be performed (using the CAST or :: syntax).</li> <li>• <b>i</b>: Implicit conversion can be performed.</li> <li>• <b>a</b>: Both explicit and implicit conversion can be performed between data types.</li> </ul>
castmethod	"char"	Conversion method. <ul style="list-style-type: none"> <li>• <b>f</b>: Conversion is performed using the specified function in the <b>castfunc</b> column.</li> <li>• <b>b</b>: Binary forcible conversion rather than the specified function in the <b>castfunc</b> column is performed between data types.</li> </ul>

## 12.2.44 PG\_CLASS

PG\_CLASS records database objects and their relationship.

**Table 12-43** PG\_CLASS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
relname	name	Name of an object, such as a table, index, or view.
relnamespace	oid	OID of the namespace that contains the relationship.
reltype	oid	Data type that corresponds to the table's row type. The index is 0 because the index does not have <b>PG_TYPE</b> records.
relowner	oid	Owner of the relationship.
relam	oid	Access method used, such as B-tree, if the row is an index.
relfilenode	oid	Name of the on-disk file of this relationship ( <b>0</b> if such file does not exist).
reltablespace	oid	Tablespace in which this relationship is stored. If the value is <b>0</b> , the default tablespace in this database is used. This column is meaningless if the relationship has no on-disk file.
relpages	double precision	Size of the on-disk representation of the table in pages (of size BLCKSZ). This is only an estimate used by the optimizer.
reltuples	double precision	Number of rows in the table. This is only an estimate used by the optimizer.
relallvisible	integer	Number of pages marked as all visible in the table. This column is used by the optimizer for optimizing SQL execution. It is updated by <b>VACUUM</b> , <b>ANALYZE</b> , and a few DDL statements such as <b>CREATE INDEX</b> .
reltoastrelid	oid	OID of the TOAST table associated with the table ( <b>0</b> if no TOAST table exists). The TOAST table stores large columns "offline" in a secondary table.
reltoastidxid	oid	OID of the index for a TOAST table ( <b>0</b> for a table other than a TOAST table).

Name	Type	Description
relhasindex	boolean	Its value is <b>true</b> if this column is a table and has (or recently had) at least one index. It is set by CREATE INDEX but is not immediately cleared by DROP INDEX. If the VACUUM process detects that a table has no index, it clears the <b>relhasindex</b> column and sets the value to <b>false</b> .
relisshared	boolean	Its value is <b>true</b> if the table is shared across all databases in the entire cluster. Otherwise, the value is <b>false</b> . Only certain system catalogs (such as PG_DATABASE) are shared.
relpersistence	"char"	<ul style="list-style-type: none"> <li>● <b>p</b>: permanent table.</li> <li>● <b>u</b>: non-log table.</li> <li>● <b>t</b>: temporary table.</li> <li>● <b>g</b>: global temporary table.</li> </ul>
relkind	"char"	<ul style="list-style-type: none"> <li>● <b>r</b>: ordinary table.</li> <li>● <b>i</b>: index.</li> <li>● <b>G</b>: global secondary index.</li> <li>● <b>s</b>: sequence.</li> <li>● <b>v</b>: view.</li> <li>● <b>t</b>: TOAST table.</li> <li>● <b>f</b>: foreign table.</li> <li>● <b>m</b>: materialized view.</li> <li>● <b>e</b>: STREAM object.</li> <li>● <b>o</b>: CONTVIEW object.</li> </ul>
relnatts	smallint	Number of user columns in the relationship (excluding system columns). <a href="#">12.2.46 PG_ATTRIBUTE</a> has the same number of rows as the user columns.
relchecks	smallint	Number of check constraints in the table. For details, see the system catalog <a href="#">12.2.53 PG_CONSTRAINT</a> .
relhasoids	boolean	Its value is <b>true</b> if an OID is generated for each row of the relationship. Otherwise, the value is <b>false</b> .
relhaspkey	boolean	Its value is <b>true</b> if the table has (or once had) a primary key. Otherwise, the value is <b>false</b> .
relhasrules	boolean	Its value is <b>true</b> if the table has rules. For details, see the system catalog <a href="#">12.2.87 PG_REWRITE</a> .
relhastriggers	boolean	The value is <b>true</b> if the table has (or once had) triggers. Triggers of the table and view are recorded in the system catalog <a href="#">12.2.99 PG_TRIGGER</a> .

Name	Type	Description
relhassubclass	boolean	Its value is <b>true</b> if the table has (or once had) any inheritance child table. Otherwise, the value is <b>false</b> .
relcmprs	tinyint	Specifies whether the compression feature is enabled for the table. Note that only batch insertion triggers compression, so ordinary CRUD does not trigger compression. <ul style="list-style-type: none"> <li>• <b>0</b>: Tables that do not support compression (primarily system catalogs, on which the compression attribute cannot be modified).</li> <li>• <b>1</b>: The compression feature of the table data is NOCOMPRESS or has no specified keyword.</li> <li>• <b>2</b>: The compression feature of the table data is COMPRESS.</li> </ul>
relhasclusterkey	boolean	Specifies whether the local cluster storage is used. <ul style="list-style-type: none"> <li>• <b>true</b>: yes</li> <li>• <b>false</b>: no</li> </ul>
relrowmovement	boolean	Specifies whether row migration is allowed when the partitioned table is updated. <ul style="list-style-type: none"> <li>• <b>true</b>: Row migration is allowed.</li> <li>• <b>false</b>: Row migration is not allowed.</li> </ul>
parttype	"char"	Specifies whether the table or index has the property of a partitioned table. <ul style="list-style-type: none"> <li>• <b>p</b>: The table or index has the property of a partitioned table.</li> <li>• <b>n</b>: The table or index does not have the property of a partitioned table.</li> </ul>
relfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table. This column is used to track whether the table needs to be vacuumed to prevent transaction ID wraparound (or to allow PG_CLOG to be shrunk). The value is <b>0 (InvalidTransactionId)</b> if the relationship is not a table.  To ensure forward compatibility, this column is reserved. The <b>relfrozenxid64</b> column is added to record the information.

Name	Type	Description
relacl	aclitem[]	<p>Access permissions. For details about the ACLItem type, see <a href="#">ACLItem</a>.</p> <p>The command output of the query is as follows:  <i>user1=privs/user2</i> indicates that the permission granted by user 2 to user 1 is <b>privs</b>.  <i>=privs/user3</i> indicates that the permission granted to the <b>public</b> role by user 3 is <b>privs</b>.</p> <p>In the preceding command, user 1, user 2, and user 3 are the existing users or roles in the database, and <b>privs</b> indicates the permissions supported by the database. For details on permission descriptions, see <a href="#">Table 12-44</a>.</p>
reloptions	text[]	Table or index access method, using character strings in the format of <i>keyword=value</i> .
relreplident	"char"	<p>Identifier of a decoding column in logical decoding.</p> <ul style="list-style-type: none"> <li>• <b>d</b>: default (primary key, if any).</li> <li>• <b>n</b>: none.</li> <li>• <b>f</b>: all columns.</li> <li>• <b>i</b>: The indisreplident of the index is specified or the default index is used.</li> </ul>
relfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table. This column is used to track whether the table needs to be vacuumed to prevent transaction ID wraparound (or to allow PG_CLOG to be shrunk). The value is <b>0 (InvalidTransactionId)</b> if the relationship is not a table.
relbucket	oid	Specifies whether the current catalog contains hash bucket shards. A valid OID points to the specific shard information recorded in the <b>PG_HASHBUCKET</b> catalog. <b>NULL</b> indicates that hash bucket shards are not included.
relbucketkey	int2vector	Hash partition column information. <b>NULL</b> indicates that the column information is not included.
relminmxid	xid	All multi-transaction IDs before this one have been replaced with a transaction ID in the table. This column is used to track whether the table needs to be vacuumed in order to prevent multi-transaction IDs wraparound or to allow <b>pg_clog</b> to be shrunk. The value is <b>0 (InvalidTransactionId)</b> if the relationship is not a table.

**Table 12-44** Description of permissions

Parameter	Parameters
r	SELECT (read)
w	UPDATE (write)
a	INSERT (insert)
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
A	ALTER
P	DROP
m	COMMENT
i	INDEX
v	VACUUM
*	Authorization options for preceding permissions

## 12.2.45 PG\_COLLATION

PG\_COLLATION describes available collations, which are essentially mappings from an SQL name to OS local categories.

**Table 12-45** PG\_COLLATION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
collname	name	-	Collation name (unique per namespace and encoding)

Name	Type	Reference	Description
collnamespace	oid	oid in <a href="#">PG_NAMESPACE</a>	OID of the namespace that contains this collation
collowner	oid	oid in <a href="#">PG_AUTHID</a>	Owner of the collation
collencoding	integer	-	Encoding in which the collation is applicable, or <b>-1</b> if it works for any encoding. It is compatible with PostgreSQL.
collcollate	name	-	<b>LC_COLLATE</b> for this collation object
collctype	name	-	<b>LC_CTYPE</b> for this collation object
collpadattr	text	-	Collation padding attribute. <ul style="list-style-type: none"> <li>• <b>NULL</b>: not applicable.</li> <li>• <b>NO PAD</b>: no padding.</li> <li>• <b>PAD SPACE</b>: blank spaces padded at the end.</li> </ul>
collisdef	boolean	-	Determines whether the collation is the default collation of the character set.

## 12.2.46 PG\_CONSTRAINT

PG\_CONSTRAINT records check, primary key, unique, and foreign key constraints on tables.

**Table 12-46** PG\_CONSTRAINT columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
conname	name	Constraint name (not necessarily unique).
connamespace	oid	OID of the namespace that contains the constraint.

Name	Type	Description
contype	"char"	<ul style="list-style-type: none"> <li>• <b>c</b>: check constraint</li> <li>• <b>p</b>: primary key constraint</li> <li>• <b>u</b>: unique constraint</li> <li>• <b>t</b>: trigger constraint</li> <li>• <b>x</b>: mutual exclusion constraint</li> <li>• <b>f</b>: foreign key constraint</li> <li>• <b>s</b>: clustering constraint</li> <li>• <b>i</b>: invalid constraint</li> </ul>
condeferrable	boolean	Specifies whether the constraint is deferrable. <ul style="list-style-type: none"> <li>• <b>true</b>: yes</li> <li>• <b>false</b>: no</li> </ul>
condeferred	boolean	Specifies whether the constraint can be deferrable by default. <ul style="list-style-type: none"> <li>• <b>true</b>: yes</li> <li>• <b>false</b>: no</li> </ul>
convalidated	boolean	Specifies whether the constraint is valid. Currently, it can be set to <b>false</b> only for foreign key and check constraints. <ul style="list-style-type: none"> <li>• <b>true</b>: valid</li> <li>• <b>false</b>: invalid</li> </ul>
conrelid	oid	Table containing this constraint ( <b>0</b> if it is not a table constraint).
contypid	oid	Domain containing this constraint ( <b>0</b> if it is not a domain constraint).
conindid	oid	ID of the index associated with the constraint.
confrelid	oid	Referenced table if this constraint is a foreign key. Otherwise, the value is <b>0</b> .
confupdtype	"char"	Foreign key update action code. <ul style="list-style-type: none"> <li>• <b>a</b>: no action</li> <li>• <b>r</b>: restriction</li> <li>• <b>c</b>: cascading</li> <li>• <b>n</b>: The parameter is set to null.</li> <li>• <b>d</b>: The default value is used.</li> </ul>

Name	Type	Description
confdeltype	"char"	Foreign key deletion action code. <ul style="list-style-type: none"> <li>• <b>a</b>: no action.</li> <li>• <b>r</b>: restriction.</li> <li>• <b>c</b>: cascading.</li> <li>• <b>n</b>: The parameter is set to null.</li> <li>• <b>d</b>: The default value is used.</li> </ul>
confmatchtype	"char"	Foreign key match type. <ul style="list-style-type: none"> <li>• <b>f</b>: full match</li> <li>• <b>p</b>: partial match</li> <li>• <b>u</b>: unspecified (The NULL value can be matched if <b>f</b> is specified.)</li> </ul>
conislocal	boolean	Specifies whether the constraint is defined locally for the relation. <ul style="list-style-type: none"> <li>• <b>true</b>: yes</li> <li>• <b>false</b>: no</li> </ul>
coninhcount	integer	Number of direct inheritance parent tables that this constraint has. When the value is not <b>0</b> , the constraint cannot be deleted or renamed.
connoinherit	boolean	Specifies whether the constraint can be inherited. <ul style="list-style-type: none"> <li>• <b>true</b>: yes</li> <li>• <b>false</b>: no</li> </ul>
consoft	boolean	Specifies whether the column indicates an informational constraint. <ul style="list-style-type: none"> <li>• <b>true</b>: yes</li> <li>• <b>false</b>: no</li> </ul>
conopt	boolean	Specifies whether you can use the informational constraint to optimize the execution plan. <ul style="list-style-type: none"> <li>• <b>true</b>: yes</li> <li>• <b>false</b>: no</li> </ul>
conkey	smallint[]	Column list of the constrained control if this column is a table constraint.
confkey	smallint[]	List of referenced columns if this column is a foreign key.
confpeqop	oid[]	ID list of the equality operators for PK = FK comparisons if this column is a foreign key.

Name	Type	Description
conppeqop	oid[]	ID list of the equality operators for PK = PK comparisons if this column is a foreign key.
conffeqop	oid[]	ID list of the equality operators for FK = FK comparisons if this column is a foreign key.
conexclp	oid[]	ID list of the per-column exclusion operators if this column is an exclusion constraint.
conbin	pg_node_tree	Internal representation of the expression if this column is a check constraint.
consrc	text	Readable representation of the expression if this column is a check constraint.
conincluding	smallint[]	Not for constraint, but will be included in the attribute column of <b>INDEX</b> .

#### NOTICE

- **consrc** is not updated when referenced objects change and does not track new column names. You are advised to use `pg_get_constraintdef()` to extract the definition of a check constraint.
- **relchecks** of [12.2.51 PG\\_CLASS](#) must agree with the number of check-constraint entries found in the table for each relationship.

## 12.2.47 PG\_CONVERSION

**PG\_CONVERSION** describes encoding conversion information.

**Table 12-47** PG\_CONVERSION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
conname	name	-	Conversion name (unique within a namespace)
connamespace	oid	OID in <a href="#">PG_NAMESPACE</a>	OID of the namespace that contains this conversion
conowner	oid	OID in <a href="#">PG_AUTHID</a>	Owner of the conversion
conforencoding	integer	-	Source encoding ID
contoencoding	integer	-	Destination encoding ID

Name	Type	Reference	Description
conproc	regproc	<b>prname</b> in <b>PG_PROC</b>	Conversion procedure
condefault	boolean	-	If this is the default conversion, the value is <b>true</b> . Otherwise, the value is <b>false</b> .

## 12.2.48 PG\_DATABASE

PG\_DATABASE records information about available databases.

**Table 12-48** PG\_DATABASE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
datname	name	Database name.
datdba	oid	Owner of the database, usually the user who created it.
encoding	integer	Character encoding for the database.
datcollate	name	Sequence used by the database.
datctype	name	Character type used by the database.
datistemplate	boolean	Specifies whether the database can be used as a template database. <ul style="list-style-type: none"> <li>• <b>true</b>: allowed.</li> <li>• <b>false</b>: not allowed.</li> </ul>
datallowconn	boolean	This column is used to protect the <b>template0</b> database from being altered. <ul style="list-style-type: none"> <li>• <b>true</b>: Users can connect to the database.</li> <li>• <b>false</b>: No user can connect to the database.</li> </ul>
datconnlimit	integer	Maximum number of concurrent connections allowed on this database. The value <b>-1</b> indicates no limit.
datlastsysoid	oid	Last system OID in the database.

Name	Type	Description
datfrozenxid	xid32	Tracks whether the database needs to be vacuumed to prevent transaction ID wraparound. This column is discarded in the current version. To ensure forward compatibility, this column is reserved. The <b>datfrozenxid64</b> column is added to record the information.
dattablespace	oid	Default tablespace of the database.
datcompatibility	name	Database compatibility mode. Currently, four compatible modes are supported: PG, ORA, MYSQL, and TD.
datacl	aclitem[]	Access permission.
datfrozenxid64	xid	Tracks whether the database needs to be vacuumed to prevent transaction ID wraparound.
datminmxid	xid	All multi-transaction IDs before this one have been replaced with a transaction ID in the database. This is used to track whether the database needs to be vacuumed in order to prevent transaction IDs wraparound or to allow PG_CLOG to be shrunk. It is the minimum value of <b>relminmxid</b> in <a href="#">12.2.51 PG_CLASS</a> of all tables in the database.
dattimezone	name	Database time zone. The default is PRC.

## 12.2.49 PG\_DB\_ROLE\_SETTING

**PG\_DB\_ROLE\_SETTING** records the default values of configuration items bound to each role and data when the database is running.

**Table 12-49** PG\_DB\_ROLE\_SETTING columns

Name	Type	Description
setdatabase	oid	Database corresponding to the configuration items ( <b>0</b> if no database is specified).
setrole	oid	Role corresponding to the configuration items ( <b>0</b> if no role is specified).
setconfig	text[]	Default value of runtime configuration items. Contact an administrator to configure it.

## 12.2.50 PG\_DEFAULT\_ACL

PG\_DEFAULT\_ACL records initial permissions assigned to newly created objects.

**Table 12-50** PG\_DEFAULT\_ACL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
defaclrole	oid	ID of the role associated with the permission.
defaclnamespace	oid	Namespace associated with the permission ( <b>0</b> if no ID).
defaclobjtype	"char"	Object type of the permission. <ul style="list-style-type: none"> <li>• <b>r</b>: table or view</li> <li>• <b>S</b>: sequence</li> <li>• <b>f</b>: function</li> <li>• <b>T</b>: type</li> <li>• <b>K</b>: client master key</li> <li>• <b>k</b>: column encryption key</li> </ul>
defaclacl	aclitem[]	Access permissions that this type of object should have on creation.

## 12.2.51 PG\_DEPEND

PG\_DEPEND records the dependency between database objects. This information allows **DROP** commands to find which other objects must be dropped by **DROP CASCADE** or prevent dropping in the **DROP RESTRICT** case.

See also [PG\\_SHDEPEND](#), which performs a similar function for dependencies involving objects that are shared across a database cluster.

**Table 12-51** PG\_DEPEND columns

Name	Type	Reference	Description
classid	oid	<b>oid</b> in <a href="#">PG_CLASS</a>	OID of the system catalog where a dependent object resides
objid	oid	Any OID column	OID of the dependent object
objsubid	integer	-	Column number for a table column ( <b>objid</b> and <b>classid</b> refer to the table itself); <b>0</b> for all other object types

Name	Type	Reference	Description
refclassid	oid	oid in <a href="#">PG_CLASS</a>	OID of the system catalog where a referenced object resides
refobjid	oid	Any OID column	OID of the referenced object
refobjsubid	integer	-	Column number for a table column ( <b>refobjid</b> and <b>refclassid</b> refer to the table itself); <b>0</b> for all other object types
deptype	"char"	-	A code defining the specific semantics of this dependency

In all cases, a PG\_DEPEND entry indicates that the referenced object cannot be dropped without also dropping the dependent object. However, there are several subflavors identified by **deptype**:

- **DEPENDENCY\_NORMAL** (n): A normal relationship between separately created objects. The dependent object can be dropped without affecting the referenced object. The referenced object can only be dropped by specifying **CASCADE**, in which case the dependent object is dropped too. Example: a table column has a normal dependency on its data type.
- **DEPENDENCY\_AUTO** (a): The dependent object can be dropped separately from the referenced object, and should be automatically dropped (regardless of RESTRICT or CASCADE mode) if the referenced object is dropped. Example: a named constraint on a table is made autodependent on the table, so that it will go away if the table is dropped.
- **DEPENDENCY\_INTERNAL** (i): The dependent object was created as part of creation of the referenced object, and is only a part of its internal implementation. A DROP of the dependent object will be disallowed outright (We'll tell the user to issue a DROP against the referenced object, instead). A DROP of the referenced object will be propagated through to drop the dependent object whether CASCADE is specified or not. Example: A trigger created to enforce a foreign-key constraint is made internally dependent on the constraint's [PG\\_CONSTRAINT](#) entry.
- **DEPENDENCY\_EXTENSION** (e): The dependent object is a member of the extension of the referenced object. The dependent object can be dropped only via DROP EXTENSION on the referenced object. Functionally this dependency type acts the same as an internal dependency, but it is kept separate for clarity and to simplify GS\_DUMP.

---

**NOTICE**

The extended function is for internal use only. You are advised not to use it.

- **DEPENDENCY\_PIN** (p): There is no dependent object; this type of entry is a signal that the system itself depends on the referenced object, and so that object must never be deleted. Entries of this type are created only by **initdb**. The columns for the dependent object contain zeroes.

## 12.2.52 PG\_DESCRIPTION

**PG\_DESCRIPTION** records optional descriptions (comments) for each database object. Descriptions of many built-in system objects are provided in the initial contents of **PG\_DESCRIPTION**.

See also **PG\_SHDESCRIPTION**, which provides a similar function for descriptions involving objects that are shared across a database cluster.

**Table 12-52** PG\_DESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this description pertains to
classoid	oid	<a href="#">PG_CLASS</a> .oid	OID of the system catalog where the object appears
objsubid	integer	-	Column number for a comment on a table column ( <b>objoid</b> and <b>classoid</b> refer to the table itself); <b>0</b> for all other object types
description	text	-	Arbitrary text that serves as the description of the object

## 12.2.53 PG\_DIRECTORY

**PG\_DIRECTORY** stores directory objects added by users. You can execute the **CREATE DIRECTORY** statement to add records to this system catalog. When **enable\_access\_server\_directory** is set to **off**, only the initial user can create directory objects. When **enable\_access\_server\_directory** is set to **on**, users with the SYSADMIN permission and users inheriting the built-in role permission **gs\_role\_directory\_create** can create directory objects. Common users can access this system catalog only after being authorized.

**Table 12-53** PG\_DIRECTORY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
dirname	name	Name of a directory object
owner	oid	Owner of a directory object
dirpath	text	Directory path.
diracl	aclitem[]	Access permissions.

## 12.2.54 PG\_ENUM

PG\_ENUM contains entries showing the values and labels for each enumerated type. The internal representation of a given enumerated value is actually the OID of its associated row in PG\_ENUM.

**Table 12-54** PG\_ENUM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
enumtypeid	oid	<b>oid</b> in <b>PG_TYPE</b>	OID of <b>12.2.105 PG_TYPE</b> owning this enumerated value.
enumsortorder	real	-	Sort position of this enumerated value within its enumerated type.
enumlabel	name	-	Textual label for this enumerated value.

The OIDs for **PG\_ENUM** rows follow a special rule: even-numbered OIDs are guaranteed to be ordered in the same way as the sort ordering of their enumerated type. If two even OIDs belong to the same enumerated type, the smaller OID must have the smaller **enumsortorder** value. Odd-numbered OID values need bear no relationship to the sort order. This rule allows the enumerated comparison routines to avoid catalog lookups in many common cases. The routines that create and alter enumerated types attempt to assign even OIDs to enumerated values whenever possible.

When an enumerated type is created, its members are assigned sort-order positions from 1 to *n*. However, members added later might be given negative or fractional values of **enumsortorder**. The only requirement on these values is that they be correctly ordered and unique within each enumerated type.

## 12.2.55 PG\_EXTENSION

PG\_EXTENSION records information about the installed extensions. By default, GaussDB provides the following extensions: PLPGSQL, DIST\_FDW, FILE\_FDW, LOG\_FDW, GC\_FDW, DBLINK\_FDW, ROACH\_API, STREAMING, TSDB, DIMSEARCH, GSREDISTRIBUTE, SECURITY\_PLUGIN, GSSTAT\_PLUGIN, PKG\_DBE\_RAW, PKG\_DBE\_OUTPUT, PKG\_DBE\_UTILITY, and PKG\_DBE\_XML. This system catalog is for internal use only. You are advised not to use it.

### NOTE

DIMSEARCH is no longer supported in the current version due to specification changes. Do not use it.

**Table 12-55** PG\_EXTENSION

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
extname	name	Extension name
extowner	oid	Owner of the extension
extnamespace	oid	Namespace containing the extension's exported objects
extrelocatable	boolean	Whether the extension can be relocated to another namespace. <b>true</b> : yes; <b>false</b> : no.
extversion	text	Version number of the extension
extconfig	oid[]	Configuration information about the extension
extcondition	text[]	Filter conditions for the extension's configuration information

## 12.2.56 PG\_FOREIGN\_DATA\_WRAPPER

**PG\_FOREIGN\_DATA\_WRAPPER** records foreign-data wrapper definitions. A foreign-data wrapper is the mechanism by which external data, residing on foreign servers, is accessed.

**Table 12-56** PG\_FOREIGN\_DATA\_WRAPPER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
fdwname	name	-	Name of a foreign-data wrapper
fdwowner	oid	<a href="#">PG_AUTHID.oid</a>	Owner of the foreign-data wrapper
fdwhandler	oid	<a href="#">PG_PROC.oid</a>	References a handler function that is responsible for supplying execution routines for the foreign-data wrapper ( <b>0</b> if no handler is provided)

Name	Type	Reference	Description
fdwvalidator	oid	<a href="#">PG_PROC.oid</a>	References a validator function that is responsible for checking the validity of the options given to the foreign-data wrapper, as well as options for foreign servers and user mappings using the foreign-data wrapper. (0 if no handler is provided)
fdwacl	aclitem[]	-	Access permissions
fdwoptions	text[]	-	Foreign-data wrapper specific option, expressed in a string in the format of keyword=value

## 12.2.57 PG\_FOREIGN\_SERVER

**PG\_FOREIGN\_SERVER** records foreign server definitions. A foreign server describes a source of external data, such as a remote server. Foreign servers are accessed via foreign-data wrappers.

**Table 12-57** PG\_FOREIGN\_SERVER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
srvname	name	-	Name of a foreign server
srvowner	oid	<a href="#">PG_AUTHID.oid</a>	Owner of the foreign server
srvfdw	oid	<a href="#">PG_FOREIGN_DATA_WRAPPER.oid</a>	OID of the foreign-data wrapper on this foreign server
srvtype	text	-	Type of the server (optional)
srvversion	text	-	Version of the server (optional)
srvacl	aclitem[]	-	Access permissions
srvoptions	text[]	-	Option used for foreign servers, expressed in a string in the format of keyword=value

## 12.2.58 PG\_HASHBUCKET

**PG\_HASHBUCKET** records hash bucket information.

**Table 12-58** PG\_HASHBUCKET columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
bucketid	oid	Hash value calculated for a bucket vector. The hash value can be used to accelerate the search for a bucket vector.
bucketcnt	integer	Number of shards.
bucketmapsize	integer	Total number of shards on all DNs.
bucketref	integer	Reserved column with <b>1</b> as its default value
bucketvector	oidvector_extend	Records all bucket IDs contained in the bucket information in this row. A unique index is created in this column. Tables with the same bucket ID share the <b>PG_HASHBUCKET</b> data in the same row.

## 12.2.59 PG\_INDEX

**PG\_INDEX** records part of index information. The rest is mostly recorded in **PG\_CLASS**.

**Table 12-59** PG\_INDEX columns

Name	Type	Description
indexrelid	oid	OID of the <a href="#">12.2.51 PG_CLASS</a> entry for the index
indrelid	oid	OID of the <a href="#">12.2.51 PG_CLASS</a> entry for the table that uses the index
indnatts	smallint	Number of columns in the index
indisunique	Boolean	Specifies whether the index is unique. <ul style="list-style-type: none"><li>● <b>true</b>: The index is unique.</li><li>● <b>false</b>: The index is not unique.</li></ul>

Name	Type	Description
indisprimary	Boolean	Specifies whether the index is the primary key of the table. <ul style="list-style-type: none"> <li>● <b>true</b>: The index is the primary key of the table. <b>indisunique</b> should always be <b>true</b> when the value of this column is <b>true</b>.</li> <li>● <b>false</b>: The index is not the primary key of the table.</li> </ul>
indisexclusion	Boolean	Specifies whether the index supports exclusive constraints. <ul style="list-style-type: none"> <li>● <b>true</b>: The index supports exclusive constraints.</li> <li>● <b>false</b>: The index does not support exclusive constraints.</li> </ul>
indimmediate	Boolean	Specifies whether to check the uniqueness of data to be inserted. <ul style="list-style-type: none"> <li>● <b>true</b>: The uniqueness check is performed immediately when data is inserted.</li> <li>● <b>false</b>: The uniqueness check is not performed when data is inserted.</li> </ul>
indisclustered	Boolean	Specifies whether the table is clustered on the index. <ul style="list-style-type: none"> <li>● <b>true</b>: The table is clustered on the index.</li> <li>● <b>false</b>: The table is not clustered on the index.</li> </ul>
indisusable	Boolean	Specifies whether the index is available for insert and select operations. <ul style="list-style-type: none"> <li>● <b>true</b>: The index is available for insert and select operations.</li> <li>● <b>false</b>: The index is unavailable for insert and select operations.</li> </ul>
indisvalid	Boolean	<ul style="list-style-type: none"> <li>● <b>true</b>: The index can be used for query.</li> <li>● <b>false</b>: The index is possibly incomplete and must still be modified by INSERT or UPDATE operations, but it cannot be securely used for queries. If it is a unique index, the uniqueness property is also not <b>true</b>.</li> </ul>
indcheckxmin	Boolean	<ul style="list-style-type: none"> <li>● <b>true</b>: Queries must not use the index until the xmin of this row in PG_INDEX is lower than their <b>TransactionXmin</b>, because the table may contain broken HOT chains with incompatible rows that they can see.</li> <li>● <b>false</b>: Indexes can be used for query.</li> </ul>

Name	Type	Description
indisready	Boolean	<ul style="list-style-type: none"><li>• <b>true</b>: The index is available for inserting data.</li><li>• <b>false</b>: The index is ignored when data is inserted or modified.</li></ul>
indkey	int2vector	This is an array of <b>indnatts</b> values indicating that this index creates table columns. For example, a value of <b>1 3</b> indicates that the first and the third columns make up the index key. The value <b>0</b> in this array indicates that the corresponding index attribute is an expression over the table columns, rather than a simple column reference.
indcollation	oidvector	OID of the collation corresponding to each index column. For details, see <a href="#">PG_COLLATION</a> .
indclass	oidvector	For each column in the index key, this contains the OID of the operator class to use. See <a href="#">PG_OPCLASS</a> for details.
indoption	int2vector	Array of values that store per-column flag bits. The meaning of the bits is defined by the index's access method.
indexprs	pg_node_tree	Expression trees (in nodeToString() representation) for index attributes that are not simple column references. It is a list with one element for each zero entry in indkey. The value is null if all index attributes are simple references.
indpred	pg_node_tree	Expression tree (in nodeToString() representation) for partial index predicate. If the index is not a partial index, this column is an empty string.
indisreplident	Boolean	Specifies whether the column of this index is a decoded column of logical decoding. <ul style="list-style-type: none"><li>• <b>true</b>: The column of this index is a decoded column of logical decoding.</li><li>• <b>false</b>: The column of this index is not a decoded column of logical decoding.</li></ul>
indnkeyatts	smallint	Total number of columns in the index. The columns that exceed the value of <b>indnatts</b> are not involved in the index query.

## 12.2.60 PG\_INHERITS

**PG\_INHERITS** records information about table inheritance hierarchies. There is one entry for each direct child table in the database. Indirect inheritance can be determined by following chains of entries.

**Table 12-60** PG\_INHERITS columns

Name	Type	Reference	Description
inhrelid	oid	<a href="#">PG_CLASS.oid</a>	OID of a child table
inhparent	oid	<a href="#">PG_CLASS.oid</a>	OID of a parent table
inhseqno	integer	-	If there is more than one direct parent for a child table (multiple inheritances), this number tells the order in which the inherited columns are to be arranged. The count starts at 1.

## 12.2.61 PG\_JOB

PG\_JOB records detailed information about jobs created by users. Dedicated threads poll the system catalog PG\_JOB and trigger jobs based on scheduled job execution time, and update job status in PG\_JOB. This system catalog belongs to the Shared Relation category. All job records are visible to all databases. Common users can access this system catalog only after being authorized.

**Table 12-61** PG\_JOB columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
job_id	bigint	Job ID, which is the primary key and is unique (with a unique index).
current_postgres_pid	bigint	If the current job has been executed, the thread ID of this job is recorded. The default value is <b>-1</b> , indicating that the job has not yet been executed.
log_user	name	Username of the creator.
priv_user	name	Username of the job executor.
dbname	name	Name of the database in which the job will be executed.
node_name	name	CN on which the job will be executed.

Name	Type	Description
job_status	"char"	<p>Execution status of the current task. The default value is 's'. The options are as follows:</p> <ul style="list-style-type: none"> <li>• 'r': running</li> <li>• 's': successfully finished</li> <li>• 'f': job failed</li> <li>• 'd': disable</li> </ul> <p>If a job fails to be executed for 16 consecutive times, <b>job_status</b> is automatically set to 'd', and no more attempt will be made on this job.</p> <p>Note: When you disable a scheduled task (by setting <b>job_queue_processes</b> to 0), the thread that monitors the job execution is not started, and the job status will not be updated. You can ignore this status. Only when the scheduled task function is enabled (<b>job_queue_processes</b> is not 0), the system updates the value of this column based on the real-time job status.</p>
start_date	timestamp without time zone	Start time of the first job execution, accurate to millisecond
next_run_date	timestamp without time zone	Time when a scheduled task is executed next time. The time is accurate to milliseconds.
failure_count	smallint	Number of times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
interval	text	Job execution interval.
last_start_date	timestamp without time zone	Start time of the last job execution, accurate to millisecond.
last_end_date	timestamp without time zone	End time of the last job execution, accurate to millisecond.
last_suc_date	timestamp without time zone	Start time of the last successful job execution, accurate to millisecond.
this_run_date	timestamp without time zone	Start time of the ongoing job execution, accurate to millisecond.
nspname	name	Name of the schema used for job execution.
job_name	text	Name of the DBE_SCHEDULER scheduled task.

Name	Type	Description
end_date	timestamp without time zone	Expiration time of the DBE_SCHEDULER scheduled task, accurate to millisecond.
enable	Boolean	The DBE_SCHEDULER scheduled task enabling status. The options are as follows: <ul style="list-style-type: none"><li>• <b>true</b>: enabled</li><li>• <b>false</b>: disabled</li></ul>
failure_message	text	Error information about the latest task execution.

## 12.2.62 PG\_JOB\_PROC

PG\_JOB\_PROC records the content of each job in [12.2.69 PG\\_JOB](#), including the PL/SQL code blocks and anonymous blocks. Storing such information in the system catalog PG\_JOB and loading it to the shared memory will result in excessive memory usage. Therefore, such information is stored in a separate table and is retrieved when needed. Common users can access this system catalog only after being authorized.

**Table 12-62** PG\_JOB\_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
job_id	integer	Foreign key, which is associated with job_id in the system catalog <a href="#">12.2.69 PG_JOB</a> .
what	text	Job content, which is the program content in the DBE_SCHEDULER scheduled task.
job_name	text	Name of the DBE_SCHEDULER scheduled task or program.

## 12.2.63 PG\_LANGUAGE

PG\_LANGUAGE registers programming languages. You can use them and APIs to write functions or stored procedures.

**Table 12-63** PG\_LANGUAGE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
lanname	name	-	Language name.
lanowner	oid	<b>oid</b> in <b>PG_AUTHID</b>	Owner of the language.
lanispl	boolean	-	<ul style="list-style-type: none"><li>• <b>true</b>: user-defined language.</li><li>• <b>false</b>: internal language, for example, SQL.</li></ul> Currently, gs_dump still uses this column to determine which languages need to be dumped, but this might be replaced by a different mechanism in the future.
lanpltrusted	boolean	-	<ul style="list-style-type: none"><li>• <b>true</b>: The language is trusted, which means that it is believed not to grant access to anything outside the normal SQL execution environment.</li><li>• <b>false</b>: The language is untrusted. Only the initial user can create functions in untrusted languages.</li></ul>
lanplcallfoid	oid	<b>oid</b> in <b>PG_PROC</b>	For non-internal languages, this column references the language handler, which is a special function responsible for executing all functions that are written in the particular language.
laninline	oid	<b>oid</b> in <b>PG_PROC</b>	This column references a function responsible for executing "inline" anonymous code blocks (DO blocks). The value is <b>0</b> if inline blocks are not supported.
lanvalidator	oid	<b>oid</b> in <b>PG_PROC</b>	This column references a language validator function responsible for checking the syntax and validity of new functions when they are created. The value is <b>0</b> if no validator is provided.
lanacl	aclitem[]	-	Access permission.

## 12.2.64 PG\_LARGEOBJECT

PG\_LARGEOBJECT records data making up large objects. A large object is identified by an OID assigned when it is created. Each large object is broken into segments or "pages" small enough to be conveniently stored as rows in PG\_LARGEOBJECT. The amount of data per page is defined to be LOBLKSIZE.

This system catalog is accessible only to system administrators.

**Table 12-64** PG\_LARGEOBJECT columns

Name	Type	Reference	Description
loid	oid	oid in <a href="#">PG_LARGEOBJECT_METADATA</a>	Identifier of the large object that includes this page
pageno	integer	-	Page number of this page within its large object (counting from zero).
data	bytea	-	Data stored in the large object. This will never be more than <b>LOBLKSIZE</b> bytes and might be less.

Each row of PG\_LARGEOBJECT holds data for one page of a large object, beginning at byte offset (**pageno \* LOBLKSIZE**) within the object. The implementation allows sparse storage: pages might be missing, and might be shorter than **LOBLKSIZE** bytes even if they are not the last page of the object. Missing regions within a large object read as zeroes.

## 12.2.65 PG\_LARGEOBJECT\_METADATA

PG\_LARGEOBJECT\_METADATA records metadata associated with large objects. The actual large object data is stored in PG\_LARGEOBJECT.

**Table 12-65** PG\_LARGEOBJECT\_METADATA columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
lomowner	oid	<a href="#">PG_AUTHID</a> .oid	Owner of the large object
lomacl	aclitem[]	-	Access permissions

## 12.2.66 PG\_NAMESPACE

PG\_NAMESPACE records namespaces, that is, schema-related information. If the database object isolation attribute is enabled, users can view only the schema information that they have the permission to access.

**Table 12-66** PG\_NAMESPACE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
nspname	name	Name of a namespace
nspowner	oid	Owner of the namespace
nsptimeline	bigint	Timeline when the namespace is created on the DN. This column is for internal use and valid only on the DN.
nspacl	aclitem[]	Access permission. For details, see <a href="#">GRANT</a> and <a href="#">REVOKE</a> .
in_redistribution	"char"	Specifies whether the content is in the redistribution state.
nspcollation	oid	Default collation of the namespace (a value may exist only when <b>sql_compatibility</b> is set to 'MYSQL').

## 12.2.67 PG\_OBJECT

PG\_OBJECT records the creator, creation time, and last modification time of objects of specified types (ordinary tables, indexes, sequences, views, stored procedures, and functions).

**Table 12-67** PG\_OBJECT columns

Name	Type	Description
object_oid	oid	Object identifier.
object_type	"char"	Object type. <ul style="list-style-type: none"><li>● <b>r</b>: ordinary table</li><li>● <b>i</b>: index</li><li>● <b>s</b>: sequence</li><li>● <b>v</b>: view</li><li>● <b>p</b>: stored procedure and function</li><li>● <b>S</b>: package header</li><li>● <b>B</b>: package body</li></ul>

Name	Type	Description
creator	oid	Object owner.
ctime	timestamp with time zone	Creation time of an object.
mtime	timestamp with time zone	Last modification time of an object. The modification operations include <b>ALTER</b> , <b>GRANT</b> , and <b>REVOKE</b> .
createcsn	bigint	CSN when an object is created.
changeocsn	bigint	CSN when DDL operations are performed on a table or an index.
valid	Boolean	Validity of an object. <b>t</b> indicates valid, and <b>f</b> indicates invalid.

**NOTICE**

- Objects created or modified during database initialization (initdb) cannot be recorded. **PG\_OBJECT** does not contain these object records.
- A database upgraded to this version cannot record objects created before the upgrade. **PG\_OBJECT** does not contain these object records.
- When the preceding two types of objects are modified again, the modification time (**mtime**) is recorded. Because the creation time of the objects cannot be obtained, **ctime** is empty.
- When an object created before the upgrade is modified again, the modification time (specified by **mtime**) is recorded. When DDL operations are performed on a table or an index, the transaction commit sequence number (specified by **changeocsn**) of the transaction to which the table or index belongs is recorded. Because the creation time of the object cannot be obtained, **ctime** and **createocsn** are empty.
- The time recorded by **ctime** and **mtime** is the start time of the transaction to which the current operation belongs.
- The time of object modification due to capacity expansion is also recorded.
- **createocsn** and **changeocsn** record the transaction commit sequence number of the transaction to which the current operation belongs.
- If the statement for creating an object has an undefined object, or the referenced object is modified or deleted, the object to be created will be invalid.

## 12.2.68 PG\_OPCLASS

**PG\_OPCLASS** defines index access method operator classes.

Each operator class defines semantics for index columns of a particular data type and a particular index access method. An operator class essentially specifies that a particular operator family is applicable to a particular indexable column data type.

The set of operators from the family that are actually usable with the indexed column are whichever ones accept the column's data type as their left-hand input.

**Table 12-68** PG\_OPCLASS columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
opcmethod	oid	OID in <a href="#">PG_AM</a>	Index access method operator class served by an operator class
opcname	name	-	Name of the operator class
opcnamespace	oid	OID in <a href="#">PG_NAMESPACE</a>	Namespace of the operator class
opcowner	oid	OID in <a href="#">PG_AUTHID</a>	Owner of the operator class
opcfamily	oid	OID in <a href="#">PG_OPFAMILY</a>	Operator family containing the operator class
opcintype	oid	OID in <a href="#">PG_TYPE</a>	Data type that the operator class indexes
opcdefault	boolean	-	The value is <b>true</b> if this operator class is the default for <b>opcintype</b> .
opckeytype	oid	OID in <a href="#">PG_TYPE</a>	Type of data stored in index, or zero if same as <b>opcintype</b>

An operator class's **opcmethod** must match the **opfmetho**d of its containing operator family.

## 12.2.69 PG\_OPERATOR

PG\_OPERATOR records information about operators.

**Table 12-69** PG\_OPERATOR columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
oprname	name	-	Name of an operator
oprnamespace	oid	<b>oid</b> in <a href="#">PG_NAMESPACE</a>	OID of the namespace that contains the operator.
oprowner	oid	<b>oid</b> in <a href="#">PG_AUTHID</a>	Owner of the operator.

Name	Type	Reference	Description
oprkind	"char"	-	<ul style="list-style-type: none"><li>• <b>b</b>: infix (both sides)</li><li>• <b>l</b>: prefix (left side)</li><li>• <b>r</b>: suffix (right side)</li></ul>
oprcanmerge	boolean	-	Whether the operator supports merge joins <ul style="list-style-type: none"><li>• <b>t</b> (true): yes</li><li>• <b>f</b> (false): no</li></ul>
oprcanhash	boolean	-	Whether the operator supports hash joins <ul style="list-style-type: none"><li>• <b>t</b> (true): yes</li><li>• <b>f</b> (false): no</li></ul>
oprleft	oid	<b>oid</b> in <b>PG_TYPE</b>	Type of the left operand.
oprright	oid	<b>oid</b> in <b>PG_TYPE</b>	Type of the right operand.
oprresult	oid	<b>oid</b> in <b>PG_TYPE</b>	Type of the result.
oprcom	oid	<b>oid</b> in <b>PG_OPERATOR</b>	<ul style="list-style-type: none"><li>• If it exists, the value is the exchange character of this operator.</li><li>• If it does not exist, the value is <b>0</b>.</li></ul>
oprnegate	oid	<b>oid</b> in <b>PG_OPERATOR</b>	<ul style="list-style-type: none"><li>• If it exists, the value is the inverter of this operator.</li><li>• If it does not exist, the value is <b>0</b>.</li></ul>
oprcode	regproc	<b>prname</b> in <b>PG_PROC</b>	Function that implements the operator.
oprrest	regproc	<b>prname</b> in <b>PG_PROC</b>	Restriction selectivity estimation function for the operator.
oprjoin	regproc	<b>prname</b> in <b>PG_PROC</b>	Join selectivity estimation function for the operator.

## 12.2.70 PG\_OPFAMILY

**PG\_OPFAMILY** defines operator families.

Each operator family is a collection of operators and associated support routines that implement semantics specified for a particular index access method. Furthermore, the operators in a family are all compatible, in a way that is specified by the access method. The operator family allows cross-data-type

operators to be used with indexes and to be reasoned about using knowledge of access method semantics.

**Table 12-70** PG\_OPFAMILY columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
opfmethod	oid	<a href="#">PG_AM.oid</a>	Index access method used by an operator family
opfname	name	-	Name of the operator family
opfnamespace	oid	<a href="#">PG_NAMESPACE.oid</a>	Namespace of the operator family
opfowner	oid	<a href="#">PG_AUTHID.oid</a>	Owner of the operator family

The majority of the information defining an operator family is not in its **PG\_OPFAMILY** row, but in the associated rows in [PG\\_AMOP](#), [PG\\_AMPROC](#), and [PG\\_OPCLASS](#).

## 12.2.71 PG\_PARTITION

PG\_PARTITION records all partitioned tables, table partitions, and index partitions in the database. Partitioned index information is not stored in the system catalog PG\_PARTITION. The partitioned table does not have actual physical files. Therefore, pg\_partition does not record information such as **relfilenode**, **relpages**, **reltuples**, **reltoastrelid**, and **reltoastidxid**.

**Table 12-71** PG\_PARTITION columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
relname	name	Names of the partitioned tables, table partitions, TOAST tables on table partitions, and index partitions.
parttype	"char"	Object type. <ul style="list-style-type: none"> <li>● <b>r</b>: partitioned table</li> <li>● <b>p</b>: table partition</li> <li>● <b>x</b>: index partition</li> </ul>

Name	Type	Description
parentid	oid	OID of the partitioned table in PG_CLASS when the object is a partitioned table or table partition. OID of the partitioned index when the object is an index partition.
rangenum	integer	Reserved column.
intervalnum	integer	Reserved column.
partstrategy	"char"	Partition policy of the partitioned table. <ul style="list-style-type: none"> <li>• 'r': range partition</li> <li>• 'l': list partition</li> <li>• 'h': hash partition</li> </ul>
relfilenode	oid	Physical storage locations of the table partition, index partition, and TOAST table on the table partition.
reltablespace	oid	OID of the tablespace containing the table partition, index partition, and TOAST table on the table partition.
relpages	double precision	Statistics: numbers of data pages of the table partition and index partition.
reltuples	double precision	Statistics: numbers of tuples of the table partition and index partition.
relallvisible	integer	Statistics: number of visible data pages of the table partition and index partition.
reltoastrelid	oid	OID of the TOAST table corresponding to the table partition.
reltoastidxid	oid	OID of the TOAST table index corresponding to the table partition.
indextblid	oid	OID of the table partition corresponding to the index partition.
indisusable	boolean	Specifies whether the index partition is available. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
relfrozenxid	xid32	Frozen transaction ID. To ensure forward compatibility, this column is reserved. The <b>relfrozenxid64</b> column is added to record the information.

Name	Type	Description
intspnum	integer	Number of tablespaces that the interval partition belongs to.
partkey	int2vector	Column number of the partition key.
intervaltablespace	oidvector	Tablespace that the interval partition belongs to. Interval partitions fall in the tablespaces in the round-robin manner.
interval	text[]	Interval value of the interval partition.
boundaries	text[]	Upper boundary of the range partition and interval partition.
transit	text[]	Transit of the interval partition.
reloptions	text[]	Storage property of a partition used for collecting online scale-out information. Same as <b>pg_class.reloptions</b> , it is expressed in a string in the format of keyword=value.
relfrozenxid64	xid	Frozen transaction ID.
relminmxid	xid	Frozen multi-transaction ID.
partitionno	integer	Used for maintaining the partition map of a partitioned table. <ul style="list-style-type: none"><li>• If the object is a partition, this field indicates the partition ID, which starts from 1 in ascending order.</li><li>• If the object is a partitioned table, this field indicates the maximum partition ID and a negative value is used for special meaning. The value increases with the DDL syntax of some partitions.</li><li>• If the object is of other types, this field is null and has no meaning.</li></ul> <b>partitionno</b> is a permanent auto-increment column, which can be reset or reclaimed by using the syntax ALTER TABLE t_name RESET PARTITION or VACUUM FULL.
subpartitionno	integer	Reserved column.

## 12.2.72 PG\_PLTEMPLATE

**PG\_PLTEMPLATE** records template information for procedural languages.

**Table 12-72** PG\_PLTEMPLATE columns

Name	Type	Description
tmplname	name	Name of the language for which this template is used
tmpltrusted	boolean	The value is <b>true</b> if the language is considered trusted. Otherwise, the value is <b>false</b> .
tmpldbcreate	boolean	The value is <b>true</b> if the language is created by the owner of the database. Otherwise, the value is <b>false</b> .
tmplhandler	text	Name of the call handler function
tmplinline	text	Name of the anonymous block handler ( <b>NULL</b> if no name of the block handler exists)
tmplvalidator	text	Name of the verification function ( <b>NULL</b> if no verification function is available)
tmpllibrary	text	Path of the shared library that implements languages
tmplacl	aclitem[]	Access permissions for template (not yet used)

## 12.2.73 PG\_PROC

PG\_PROC records information about functions or procedures.

**Table 12-73** PG\_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
proname	name	Function name.
pronamespace	oid	OID of the namespace that contains the function.
proowner	oid	Owner of the function.
prolang	oid	Implementation language or call API of the function.
procost	real	Estimated execution cost.
prorows	real	Estimated number of rows that are influenced.

Name	Type	Description
provariadic	oid	Data type of parameter element.
protransform	regproc	Simplified call method for the function.
proisagg	Boolean	Specifies whether the function is an aggregate function. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
proiswindow	Boolean	Specifies whether the function is a window function. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
prosecdef	Boolean	Specifies whether the function is a security definer (or a setuid function). <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
proleakproof	Boolean	Specifies whether the function has side effects. If no leakproof treatment is provided for parameters, the function throws errors. <ul style="list-style-type: none"> <li>• <b>t</b> (true): There is no side effect.</li> <li>• <b>f</b> (false): There are side effects.</li> </ul>
proisstrict	Boolean	<ul style="list-style-type: none"> <li>• <b>t</b> (true): When this function is used, the function does not call the function and returns null if the input parameter is null.</li> <li>• <b>f</b> (false): When this function is used, it is called even if the input parameter is null. Therefore, this function must process null input.</li> </ul>
proretset	Boolean	Specifies whether the return value of the function is a set (that is, multiple values of the specified data type). <ul style="list-style-type: none"> <li>• <b>true</b>: The return value of the function is a set.</li> <li>• <b>false</b>: The return value of the function is not a set.</li> </ul>

Name	Type	Description
provolatile	"char"	Specifies whether the function's result depends only on its input parameters, or is affected by outside factors. <ul style="list-style-type: none"> <li>• <b>i</b>: immutable functions, which always deliver the same result for the same inputs.</li> <li>• <b>s</b>: stable functions, whose results (for fixed inputs) do not change within a scan.</li> <li>• <b>v</b>: volatile functions, whose results may change at any time. Use <b>v</b> also for functions with side-effects, so that the engine cannot get optimized if volatile functions are called.</li> </ul>
pronargs	smallint	Number of parameters.
pronargdefaults	smallint	Number of parameters that have default values.
prorettype	oid	Data type of return values.
proargtypes	oidvector	Array that stores the data types of function parameters. This array includes only input parameters (including <b>INOUT</b> parameters), and indicates the call signature (API) of the function.
proallargtypes	oid[]	Array that contains the data types of function parameters. This array includes all parameter types (including <b>OUT</b> and <b>INOUT</b> parameters); however, if all the parameters are <b>IN</b> parameters, this column is null. Note that array indexing is 1-based, whereas for historical reasons, <b>proargtypes</b> is indexed from 0.
proargmodes	"char"[]	Array with the modes of the function parameters, encoded as follows: <ul style="list-style-type: none"> <li>• <b>i</b> indicates the IN parameter.</li> <li>• <b>o</b> indicates the OUT parameter.</li> <li>• <b>b</b> indicates the INOUT parameter.</li> <li>• <b>v</b> indicates the VARIADIC parameter.</li> </ul> If all the parameters are <b>IN</b> parameters, this column is null. Note that indexes correspond to positions of <b>proallargtypes</b> , not <b>proargtypes</b> .
proargnames	text[]	Array that stores the names of the function parameters. Parameters without a name are set to empty strings in the array. If none of the parameters have a name, this column is null. Note that indexes correspond to positions of <b>proallargtypes</b> , not <b>proargtypes</b> .

Name	Type	Description
proargdefaults	pg_node_tree	Expression tree of the default value. This is the list of <b>pronargdefaults</b> elements.
prosrc	text	A definition that describes a function or stored procedure. In an interpreting language, it is the function source code, a connection symbol, a file name, or any body content specified when a function or stored procedure is created, depending on how a language or call is used.
probin	text	Additional information about how to call the function. Again, the interpretation is language-specific.
proconfig	text[]	Function's local settings for run-time configuration variables.
proacl	aclitem[]	Access permission. For details, see <b>GRANT</b> and <b>REVOKE</b> .
prodefaultargpos	int2vector	Position of the input parameter of a function with a default value.
fencedmode	Boolean	Execution mode of a function, indicating whether the function is executed in fence or not fence mode. <ul style="list-style-type: none"> <li>● <b>true</b>: fence mode. In this mode, the function is executed in the fork process.</li> <li>● <b>false</b>: not fence mode.</li> </ul>
proshippable	Boolean	Specifies whether the function can be pushed down to DN for execution. The default value is <b>false</b> . <ul style="list-style-type: none"> <li>● Functions of the <b>IMMUTABLE</b> type can always be pushed down to DN.</li> <li>● Functions of the <b>STABLE</b> or <b>VOLATILE</b> type can be pushed down to DN only if their attribute is <b>SHIPPABLE</b>.</li> </ul>
propackage	Boolean	Specifies whether the function supports overloading. The default value is <b>false</b> . <ul style="list-style-type: none"> <li>● <b>t</b> (true): supported.</li> <li>● <b>f</b> (false): not supported.</li> </ul>
prokind	"char"	Specifies whether the object is a function or a stored procedure. <ul style="list-style-type: none"> <li>● <b>'f'</b>: The object is a function.</li> <li>● <b>'p'</b>: The object is a stored procedure.</li> </ul>

Name	Type	Description
proargsrc	text	Describes the parameter input strings of functions or stored procedures that are compatible with ORA syntax, including parameter comments. The default value is <b>NULL</b> .
proargtypesext	oidvector_ _extend	Data type array used to store function parameters when there are a large number of function parameters. This array includes only input parameters (including <b>INOUT</b> parameters), and indicates the call signature (API) of the function.
prodefaultargpo- sext	int2vector_ _extend	Position of the input parameter with a default value when the function has a large number of parameters.
allargtypes	oidvector	Array for storing the data types of stored procedure parameters, including all parameters (input parameters, output parameters, and <b>INOUT</b> parameters) of the stored procedure.
allargtypesext	oidvector_ _extend	Array for storing the data types of stored procedure parameters when the number of function parameters is greater than 666. The array contains all parameters (including input parameters, output parameters, and <b>INOUT</b> parameters).

 **NOTE**

When a function is created, data is inserted into the PG\_PROC catalog to update the index. When there are a large number of input and output parameters, the index length may exceed one-third of the page length. As a result, the error "Index row size xxx exceeds maximum xxx for index 'pg\_proc\_proname\_all\_args\_nsp\_index'" may be reported as expected. You can reduce the number of parameters to avoid this error.

## 12.2.74 PG\_RANGE

PG\_RANGE records information about range types, except for records of types in [PG\\_TYPE](#).

**Table 12-74** PG\_RANGE columns

Name	Type	Reference	Description
rngtypeid	oid	<b>oid</b> in <a href="#">PG_TYPE</a>	OID of the range type
rngcollation	oid	<b>oid</b> in <a href="#">PG_COLLATION</a>	OID of the collation used for range comparisons ( <b>0</b> if none)

Name	Type	Reference	Description
rngsubopc	oid	oid in <a href="#">PG_OPCLASS</a>	OID of the subtype's operator class used for range comparisons
rngcanonical	regproc	proname in <a href="#">PG_PROC</a>	Name of the function to convert a range value into canonical form ( <b>0</b> if none)
rngsubdiff	regproc	proname in <a href="#">PG_PROC</a>	Name of the function used to calculate the difference between two elements in a range. The return value of the function is of the double-precision type. If the function does not exist, the value of <b>rngsubdiff</b> is <b>0</b> .

**rngsubopc** determines the sort ordering used by the range type. **rngcanonical** is used when the element type is discrete.

## 12.2.75 PG\_REPLICATION\_ORIGIN

PG\_REPLICATION\_ORIGIN contains all created replication sources and is shared among all databases in a cluster. Each cluster has only one copy of PG\_REPLICATION\_ORIGIN, not one copy per database instance.

**Table 12-75** PG\_REPLICATION\_ORIGIN columns

Name	Type	Description
roident	oid	Unique replication source identifier within a cluster.
roname	text	External user-defined replication source name.

## 12.2.76 PG\_RESOURCE\_POOL

PG\_RESOURCE\_POOL provides information about database resource pools.

**Table 12-76** PG\_RESOURCE\_POOL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).

Name	Type	Description
respool_name	name	Name of a resource pool.
mem_percent	integer	Percentage of the memory configuration.
cpu_affinity	bigint	Value of cores bound to the CPU.
control_group	name	Name of the Cgroup where the resource pool is located.
max_dop	integer	Maximum degree of concurrency. This column is used for scaling and indicates the scanning concurrency during data redistribution.
memory_limit	name	Maximum memory of the resource pool.
parentid	oid	OID of the parent resource pool.
io_limits	integer	Upper limit of IOPS. It is counted by 10 thousands per second.
io_priority	name	I/O priority set for jobs that consume many I/O resources. It takes effect when the I/O usage reaches 90%.
max_worker	integer	Concurrency in a table during data redistribution. This column is used only for scaling.
max_connections	integer	Maximum number of connections that can be used by a resource pool.

## 12.2.77 PG\_REWRITE

PG\_REWRITE records rewrite rules defined for tables and views.

**Table 12-77** PG\_REWRITE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
rulename	name	Name of a rule.
ev_class	oid	Name of the table that uses the rule.
ev_attr	smallint	Column to which this rule applies (always <b>0</b> to indicate the entire table).

Name	Type	Description
ev_type	"char"	Event type for the rule. <ul style="list-style-type: none"> <li>• 1: SELECT</li> <li>• 2: UPDATE</li> <li>• 3: INSERT</li> <li>• 4: DELETE</li> </ul>
ev_enabled	"char"	Controls the mode in which the rule is triggered. <ul style="list-style-type: none"> <li>• <b>O</b>: The rule is triggered in origin and local modes.</li> <li>• <b>D</b>: The rule is disabled.</li> <li>• <b>R</b>: The rule is triggered in replica mode.</li> <li>• <b>A</b>: The rule is always triggered.</li> </ul>
is_instead	Boolean	The value is <b>true</b> if the rule is of the <b>INSTEAD</b> type. Otherwise, the value is <b>false</b> .
ev_qual	pg_node_tree	Expression tree (in the form of a <code>nodeToString()</code> representation) for the rule's qualifying condition
ev_action	pg_node_tree	Query tree (in the form of a <code>nodeToString()</code> representation) for the rule's action

## 12.2.78 PG\_RLSPOLICY

PG\_RLSPOLICY records row-level security policies.

**Table 12-78** PG\_RLSPOLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
polname	name	Name of an access control policy.
polrelid	oid	OID of the table object on which the row-level security policy takes effect.
polcmd	"char"	SQL operations affected by the row-level security policy.
polpermissive	Boolean	Attribute of the row-level security policy. <ul style="list-style-type: none"> <li>• <b>t</b>: OR condition concatenation of expressions.</li> <li>• <b>f</b>: AND condition concatenation of expressions.</li> </ul>

Name	Type	Description
polroles	oid[]	OID list of users affected by the row-level access control policy. If this parameter is not specified, all users are affected.
polqual	pg_node_tree	Expression of a row-level security policy.

## 12.2.79 PG\_SECLABEL

**PG\_SECLABEL** records security labels on database objects.

See also [PG\\_SHSECLABEL](#), which provides a similar function for security labels of database objects that are shared across a database cluster.

**Table 12-79** PG\_SECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to
classoid	oid	<a href="#">PG_CLASS</a> .oid	OID of the system catalog where the object appears
objsubid	integer	-	Column number for a security label on a table column
provider	text	-	Label provider associated with the label
label	text	-	Security label applied to the object

## 12.2.80 PG\_SET

**PG\_SET** records metadata defined by the SET data type. Currently, this function is not supported in distributed mode.

## 12.2.81 PG\_SHDEPEND

**PG\_SHDEPEND** records the dependency between database objects and shared objects, such as roles. Based on this information, GaussDB can ensure that those objects are unreferenced before attempting to delete them.

See also [PG\\_DEPEND](#), which provides a similar function for dependencies involving objects within a single database.

**PG\_SHDEPEND** is shared among all databases of cluster. That is, there is only one **PG\_SHDEPEND** for each cluster, not for each database.

**Table 12-80** PG\_SHDEPEND columns

Name	Type	Reference	Description
dbid	oid	<b>oid</b> in <a href="#">PG_DATABASE</a>	OID of the database the dependent object is in, or zero for a shared object.
classid	oid	<b>oid</b> in <a href="#">PG_CLASS</a>	OID of the system catalog where a dependent object resides.
objid	oid	Any OID column	OID of the dependent object
objsubid	integer	-	Column number for a table column ( <b>objid</b> and <b>classid</b> refer to the table itself); <b>0</b> for all other object types
refclassid	oid	<b>oid</b> in <a href="#">PG_CLASS</a>	OID of the system catalog the referenced object is in (must be a shared catalog).
refobjid	oid	Any OID column	OID of the referenced object
deptype	"char"	-	Code segment defining the specific semantics of this dependency relationship. See the following for details.

In all cases, a PG\_SHDEPEND entry indicates that the referenced object cannot be dropped without also dropping the dependent object. However, there are several subflavors identified by **deptype**:

- SHARED\_DEPENDENCY\_OWNER (o)  
The referenced object (which must be a role) is the owner of the dependent object.
- SHARED\_DEPENDENCY\_ACL (a)  
The referenced object (which must be a role) is mentioned in the access control list (ACL) of the dependent object. SHARED\_DEPENDENCY\_ACL does not add a record to the owner of the object because the owner will have a SHARED\_DEPENDENCY\_OWNER record.
- SHARED\_DEPENDENCY\_PIN (p)  
This type of record indicates that the system itself depends on the depended object. Therefore, such an object cannot be deleted. Entries of this type are created only by initdb. The columns for the dependent object contain zeroes.
- SHARED\_DEPENDENCY\_DBPRIV(d)  
The referenced object (must be a role) has the ANY permission on the dependent object (the specified OID of the dependent object corresponds to a row in the [12.2.11 GS\\_DB\\_PRIVILEGE](#) system catalog).

## 12.2.82 PG\_SHDESCRIPTION

PG\_SHDESCRIPTION records optional comments for shared database objects. Descriptions can be manipulated with the **COMMENT** command and viewed with the `\d` command.

See also PG\_DESCRIPTION, which provides a similar function for descriptions involving objects within a single database.

PG\_SHDESCRIPTION is shared among all databases of cluster. That is, there is only one PG\_SHDESCRIPTION for each cluster, not for each database.

**Table 12-81** PG\_SHDESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the described object.
classoid	oid	<b>oid</b> in <a href="#">PG_CLASS</a>	OID of the system catalog where the object appears.
description	text	-	Description of the object.

## 12.2.83 PG\_SHSECLABEL

PG\_SHSECLABEL records security labels on shared database objects. Security labels can be manipulated with the **SECURITY LABEL** command.

For an easier way to view security labels, see [PG\\_SECLABELS](#).

See also [PG\\_SECLABEL](#), which provides a similar function for security labels involving objects within a single database.

Unlike most system catalogs, **PG\_SHSECLABEL** is shared across all databases of a cluster. There is only one copy of **PG\_SHSECLABEL** per cluster, not one per database.

**Table 12-82** PG\_SHSECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to
classoid	oid	<a href="#">PG_CLASS.oid</a>	OID of the system catalog where the object appears
provider	text	-	Label provider associated with the label
label	text	-	Security label applied to the object

## 12.2.84 PG\_STATISTIC

PG\_STATISTIC records statistics about tables and index columns in a database. By default, only system administrators can access the system catalog. Common users can access the system catalog only after being authorized.

**Table 12-83** PG\_STATISTIC columns

Name	Type	Description
starelid	oid	Table or index that the described column belongs to.
starelkind	"char"	Type of an object.
staattnum	smallint	Number of the described column in the table, starting from 1.
stainherit	boolean	Specifies whether to collect statistics for objects that have inheritance relationship.
stanullfrac	real	Percentage of column entries that are null.
stawidth	integer	Average stored width, in bytes, of non-null entries.
stadistinct	real	Number of distinct, non-null data values in the column for all DNs. <ul style="list-style-type: none"><li>• A value greater than 0 is the actual number of distinct values.</li><li>• A value less than 0 is the ratio of the <b>distinct</b> value to the total number of rows. For example, if <b>stadistinct</b> is <b>-0.5</b>, the actual <b>distinct</b> value is the total number of rows multiplied by 0.5.</li><li>• The value <b>0</b> indicates that the number of distinct values is unknown.</li></ul>
stakindN	smallint	Code number stating that the type of statistics is stored in slot <i>N</i> of the <i>pg_statistic</i> row. The value of <i>N</i> ranges from 1 to 5.
staopN	oid	Operator used to generate the statistics stored in slot <i>N</i> . For example, a histogram slot shows the < operator that defines the sort order of the data. The value of <i>N</i> ranges from 1 to 5.
stanumbers N	real[]	Numerical statistics of the appropriate type for slot <i>N</i> . The value is <b>NULL</b> if the slot does not involve numerical values. The value of <i>N</i> ranges from 1 to 5.

Name	Type	Description
stavaluesN	anyarray	Column data values of the appropriate type for slot <i>N</i> . The value is <b>NULL</b> if the slot type does not store any data values. Each array's element values are actually of the specific column's data type so there is no way to define these columns' type more specifically than anyarray. The value of <i>N</i> ranges from 1 to 5.
stadndistinct	real	Number of unique non-null data values in the <b>DN1</b> column. <ul style="list-style-type: none"> <li>• A value greater than 0 is the actual number of distinct values.</li> <li>• A value less than 0 is the ratio of the <b>distinct</b> value to the total number of rows. For example, if <b>stadndistinct</b> is <b>-0.5</b>, the actual <b>distinct</b> value is the total number of rows multiplied by 0.5.</li> <li>• The value <b>0</b> indicates that the number of distinct values is unknown.</li> </ul>
staextinfo	text	Information about extension statistics. This is reserved.

**NOTICE**

PG\_STATISTIC stores sensitive information about statistical objects, such as MCVs. System administrators and authorized users can access the PG\_STATISTIC system catalog to query the sensitive information about the statistical objects.

## 12.2.85 PG\_STATISTIC\_EXT

PG\_STATISTIC\_EXT displays extended statistics of tables in a database, such as statistics of multiple columns. Statistics of expressions will be supported later. You can specify the extended statistics to be collected. This system catalog is accessible only to system administrators.

**Table 12-84** PG\_STATISTIC\_EXT columns

Name	Type	Description
starelid	oid	Table or index that the described column belongs to.
starelkind	char	Type of an object. 'c' indicates a table, and 'p' indicates a partition.

Name	Type	Description
stainherit	boolean	Specifies whether to collect statistics for objects that have inheritance relationship. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
stanullfrac	real	Percentage of column entries that are null.
stawidth	integer	Average stored width, in bytes, of non-null entries.
stadistinct	real	Number of distinct, non-null data values in the column for all DNs. <ul style="list-style-type: none"> <li>• A value greater than 0 is the actual number of distinct values.</li> <li>• A value less than 0 is the ratio of the <b>distinct</b> value to the total number of rows. For example, if <b>stadistinct</b> is <b>-0.5</b>, the actual <b>distinct</b> value is the total number of rows multiplied by 0.5.</li> <li>• The value <b>0</b> indicates that the number of distinct values is unknown.</li> </ul>
stadndistinct	real	Number of unique non-null data values in the <b>DN1</b> column. <ul style="list-style-type: none"> <li>• A value greater than 0 is the actual number of distinct values.</li> <li>• A value less than 0 is the ratio of the <b>distinct</b> value to the total number of rows. For example, if <b>stadndistinct</b> is <b>-0.5</b>, the actual <b>distinct</b> value is the total number of rows multiplied by 0.5.</li> <li>• The value <b>0</b> indicates that the number of distinct values is unknown.</li> </ul>
stakindN	smallint	Code number stating that the type of statistics is stored in slot <i>N</i> of the <i>pg_statistic</i> row. The value of <i>N</i> ranges from 1 to 5.
staopN	oid	Operator used to generate the statistics stored in slot <i>N</i> . For example, a histogram slot shows the < operator that defines the sort order of the data. The value of <i>N</i> ranges from 1 to 5.
stakey	int2vector	Array of a column ID.
stanumbers N	real[]	Numerical statistics of the appropriate type for slot <i>N</i> . The value is <b>NULL</b> if the slot does not involve numerical values. The value of <i>N</i> ranges from 1 to 5.

Name	Type	Description
stavaluesN	anyarray	Column data values of the appropriate type for slot <i>N</i> . The value is <b>NULL</b> if the slot type does not store any data values. Each array's element values are actually of the specific column's data type so there is no way to define these columns' type more specifically than anyarray. The value of <i>N</i> ranges from 1 to 5.
staexprs	pg_node_tree	Expression corresponding to the extended statistics information.
stasource	char	Source of extended statistics: <ul style="list-style-type: none"> <li>'a': indicates automatic creation. For details, see the GUC parameter <b>auto_statistic_ext_columns</b>.</li> <li>'m': indicates that a user manually creates the statistics data using analyze tablename ((column list)) or alter table tablename add statistics ((column list)).</li> </ul>
stastatus	char	Status of extended statistics: <ul style="list-style-type: none"> <li>'a': active and available.</li> <li>'d': disabled. Related information is not collected, and the optimizer does not use the data when generating a plan. You can use the alter table tablename disable/enable statistics((column list)) syntax to modify the status of extended statistics.</li> </ul>

#### NOTICE

PG\_STATISTIC\_EXT stores sensitive information about statistical objects, such as MCVs. System administrators and authorized users can access the PG\_STATISTIC\_EXT system catalog to query the sensitive information about the statistical objects.

## 12.2.86 PG\_SYNONYM

PG\_SYNONYM records the mapping between synonym object names and other database object names.

**Table 12-85** PG\_SYNONYM columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
synname	name	Synonym name.

Name	Type	Description
synnamespace	oid	OID of the namespace that contains the synonym.
synowner	oid	Owner of the synonym, usually the OID of the user who created it.
synobjschema	name	Schema name specified by the associated object.
synobjname	name	Name of the associated object.
syndblinkname	name	Database link name.

## 12.2.87 PG\_TABLESPACE

**PG\_TABLESPACE** records tablespace information.

**Table 12-86** PG\_TABLESPACE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
spcname	name	Tablespace name
spcowner	oid	Owner of the tablespace, usually the user who created it
spcacl	aclitem[]	Access permissions. For details, see <a href="#">GRANT</a> and <a href="#">REVOKE</a> .
spcoptions	text[]	Options of the tablespace
spcmaxsize	text	Maximum size of the available disk space, in bytes
relative	boolean	Whether the storage path specified by the tablespace is a relative path <ul style="list-style-type: none"><li>• <b>t (true)</b>: yes.</li><li>• <b>f (false)</b>: no.</li></ul>

## 12.2.88 PG\_TRIGGER

**PG\_TRIGGER** records trigger information.

**Table 12-87** PG\_TRIGGER columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
tgrelid	oid	OID of the table where the trigger is located.
tgname	name	Trigger name.
tgfoid	oid	Function to be invoked by a trigger.
tgtype	smallint	Trigger type.
tgenabled	"char"	<b>O</b> : The trigger is triggered in origin or local mode. <b>D</b> : The trigger is disabled. <b>R</b> : The trigger is triggered in replica mode. <b>A</b> : The trigger is always triggered.
tgisinternal	Boolean	Internal trigger ID. If the value is <b>true</b> , it indicates an internal trigger.
tgconstrrelid	oid	Table referenced by the integrity constraint.
tgconstrindid	oid	Index of the integrity constraint.
tgconstraint	oid	OID of the constraint trigger in PG_CONSTRAINT.
tgdeferrable	Boolean	Whether the constraint trigger is of the DEFERRABLE type <ul style="list-style-type: none"><li>• <b>t (true)</b>: yes.</li><li>• <b>f (false)</b>: no.</li></ul>
tginitdeferred	Boolean	Specifies whether the trigger is of the INITIALLY DEFERRED type. <ul style="list-style-type: none"><li>• <b>t (true)</b>: yes</li><li>• <b>f (false)</b>: no</li></ul>
tgargs	smallint	Number of input parameters of the trigger function.
tgattr	int2vector	Column ID specified by the trigger. If no column is specified, an empty array is used.
tgargs	bytea	Parameter transferred to the trigger.
tgqual	pg_node_tree	WHEN condition of the trigger ( <b>NULL</b> if the WHEN condition does not exist).
tgowner	oid	Trigger owner.

## 12.2.89 PG\_TS\_CONFIG

**PG\_TS\_CONFIG** contains entries representing text search configurations. A configuration specifies a particular text search parser and a list of dictionaries to use for each of the parser's output token types.

The parser is shown in the **PG\_TS\_CONFIG** entry, but the token-to-dictionary mapping is defined by subsidiary entries in **PG\_TS\_CONFIG\_MAP**.

**Table 12-88** PG\_TS\_CONFIG columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
cfgname	name	-	Text search configuration name
cfgnamespace	oid	<a href="#">PG_NAMESPACE.oid</a>	OID of the namespace that contains the configuration
cfgowner	oid	<a href="#">PG_AUTHID.oid</a>	Owner of the configuration
cfgparser	oid	<a href="#">PG_TS_PARSER.oid</a>	OID of the text search parser for this configuration
cfoptions	text[]	-	Configuration options

## 12.2.90 PG\_TS\_CONFIG\_MAP

**PG\_TS\_CONFIG\_MAP** contains entries showing which text search dictionaries should be consulted, and in what order, for each output token type of each text search configuration's parser.

**Table 12-89** PG\_TS\_CONFIG\_MAP columns

Name	Type	Reference	Description
mapcfg	oid	<a href="#">PG_TS_CONFIG.oid</a>	OID of the <a href="#">PG_TS_CONFIG</a> entry owning this map entry
maptokentype	integer	-	Token type generated by the configuration's parser
mapseqno	integer	-	Sequence number of a token type when the values of <b>mapcfg</b> or <b>maptokentype</b> are the same.
mapdict	oid	<a href="#">PG_TS_DICT.oid</a>	OID of the text search dictionary to consult

## 12.2.91 PG\_TS\_DICT

**PG\_TS\_DICT** contains entries that define text search dictionaries. A dictionary depends on a text search template, which specifies all the implementation functions needed; the dictionary itself provides values for the user-settable parameters supported by the template.

This division of labor allows dictionaries to be created by unprivileged users. The parameters are specified by a text string **dictinitoption**, whose format and meaning vary depending on the template.

**Table 12-90** PG\_TS\_DICT columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
dictname	name	-	Text search dictionary name
dictnamespace	oid	<a href="#">PG_NAMESPACE.oid</a>	OID of the namespace that contains the dictionary
dictowner	oid	<a href="#">PG_AUTHID.oid</a>	Owner of the dictionary
dicttemplate	oid	<a href="#">PG_TS_TEMPLATE.oid</a>	OID of the text search template for the dictionary
dictinitoption	text	-	Initialization option string for the template

## 12.2.92 PG\_TS\_PARSER

**PG\_TS\_PARSER** contains entries defining text search parsers. A parser is responsible for splitting input text into lexemes and assigning a token type to each lexeme. Since a parser must be implemented by C-language-level functions, creation of new parsers is restricted to database superusers.

**Table 12-91** PG\_TS\_PARSER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
prsname	name	-	Text search parser name
prsnamespace	oid	<a href="#">PG_NAMESPACE.oid</a>	OID of the namespace that contains the parser

Name	Type	Reference	Description
prsstart	regproc	<a href="#">PG_PROC.proname</a>	Name of the parser's startup function
prstoken	regproc	<a href="#">PG_PROC.proname</a>	Name of the parser's next-token function
prsend	regproc	<a href="#">PG_PROC.proname</a>	Name of the parser's shutdown function
prsheadline	regproc	<a href="#">PG_PROC.proname</a>	Name of the parser's headline function
prsllextype	regproc	<a href="#">PG_PROC.proname</a>	Name of the parser's lextype function

### 12.2.93 PG\_TS\_TEMPLATE

**PG\_TS\_TEMPLATE** contains entries defining text search templates. A template provides a framework for text search dictionaries. Since a template must be implemented by C-language-level functions, templates can be created only by database administrators.

**Table 12-92** PG\_TS\_TEMPLATE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
tmplname	name	-	Text search template name
tmplnamespace	oid	<a href="#">PG_NAMESPACE.oid</a>	OID of the namespace that contains the template
tmplinit	regproc	<a href="#">PG_PROC.proname</a>	Name of the template's initialization function
tmpllexize	regproc	<a href="#">PG_PROC.proname</a>	Name of the template's lexize function

### 12.2.94 PG\_TYPE

**PG\_TYPE** stores information about data types.

**Table 12-93** PG\_TYPE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
typname	name	Data type name
typnamespace	oid	OID of the namespace that contains the type.
typowner	oid	Owner of the type.
typplen	smallint	Number of bytes in the internal representation of the type for a fixed-size type. It is a negative number for a variable-length type. <ul style="list-style-type: none"> <li>• <b>-1</b>: a "varlena" type (one that has a length word).</li> <li>• <b>-2</b>: a null-terminated C string.</li> </ul>
typbyval	Boolean	<ul style="list-style-type: none"> <li>• <b>true</b>: A value of this type is transferred by value internally.</li> <li>• <b>false</b>: A value of this type is transferred by reference internally.</li> </ul> <p>If <b>typplen</b> of this type is not <b>1, 2, 4, or 8</b>, you are advised to transfer a reference value for <b>typbyval</b>. You can also transfer a value for <b>typbyval</b>. Variable-length types are usually passed by reference or by value.</p>
typtype	"char"	<ul style="list-style-type: none"> <li>• <b>b</b>: basic type.</li> <li>• <b>d</b>: domain type.</li> <li>• <b>p</b>: pseudo type.</li> <li>• <b>r</b>: range type.</li> <li>• <b>e</b>: enumeration type.</li> <li>• <b>u</b>: undefined type.</li> <li>• <b>o</b>: set type.</li> </ul> <p>For details, see <b>typrelid</b> and <b>typbasetype</b>.</p>
typcategory	"char"	Fuzzy classification of data types, which can be used by parsers as the basis for data conversion.
typispreferred	Boolean	<p><b>true</b>: Data is converted when it complies with the conversion rule specified by <b>typcategory</b>.</p> <ul style="list-style-type: none"> <li>• <b>false</b>: Data is not converted.</li> </ul>
typisdefined	Boolean	<ul style="list-style-type: none"> <li>• <b>true</b>: The type has been defined.</li> <li>• <b>false</b>: The placeholder of an undefined type is used. In this case, there is no reliable information except the name, namespace, and OID of the type.</li> </ul>

Name	Type	Description
typdelim	"char"	Character that separates two values of this type when parsing an array input. Note that the delimiter is associated with the array element data type, not the array data type.
typrelid	oid	<b>0</b> .
typelem	oid	If <b>typelem</b> is not <b>0</b> , it identifies another row in PG_TYPE. The current type can be described as an array yielding values of type <b>typelem</b> . A "true" array type has a variable length ( <b>typlen</b> = <b>-1</b> ), but some fixed-length types ( <b>typlen</b> > <b>0</b> ) also have non-zero <b>typelem</b> , for example <b>name</b> and <b>point</b> . If a fixed-length type has a <b>typelem</b> , its internal representation must be a number of values of the <b>typelem</b> data type with no other data. Variable-length array types have a header defined by the array subroutines.
typarray	oid	If the value is not <b>0</b> , the corresponding type record is available in PG_TYPE.
typinput	regproc	Input conversion function (text format).
typoutput	regproc	Output conversion function (text format).
typreceive	regproc	Input conversion function (binary format); <b>0</b> for non-input conversion function.
typsend	regproc	Output conversion function (binary format); <b>0</b> for non-output conversion function.
typmodin	regproc	Input type modifier function; <b>0</b> if none.
typmodout	regproc	Output type modifier function; <b>0</b> if none.
typanalyze	regproc	Custom ANALYZE function; <b>0</b> if the standard function is used.

Name	Type	Description
typalign	"char"	<p>Alignment required when storing a value of this type. It applies to storage on disks as well as most representations of the value. When multiple values are stored consecutively, such as in the representation of a complete row on disk, padding is inserted before a data of this type so that it begins on the specified boundary. The alignment reference is the beginning of the first datum in the sequence. Possible values are:</p> <ul style="list-style-type: none"> <li>• <b>c</b>: char alignment, that is, no alignment required</li> <li>• <b>s</b>: short alignment (2 bytes on most machines)</li> <li>• <b>i</b>: integer alignment (4 bytes on most machines)</li> <li>• <b>d</b>: double alignment (8 bytes on many machines, but by no means all)</li> </ul> <p><b>NOTICE</b> For types used in system catalogs, the size and alignment defined in PG_TYPE must agree with the way that the compiler lays out the column in a structure representing a table row.</p>
typstorage	"char"	<p><b>typstorage</b> tells for varlena types (those with <b>typlen = -1</b>) if the type is prepared for toasting and what the default strategy for attributes of this type should be. Possible values are:</p> <ul style="list-style-type: none"> <li>• <b>p</b>: Values are always stored plain.</li> <li>• <b>e</b>: Value can be stored in a secondary relationship (if the relation has one, see <b>reltoastrelid</b> in <a href="#">12.2.51 PG_CLASS</a>).</li> <li>• <b>m</b>: Values can be stored compressed inline.</li> <li>• <b>x</b>: Values can be stored compressed inline or stored in secondary storage.</li> </ul> <p><b>NOTICE</b> <b>m</b> domains can also be moved out to secondary storage, but only as a last resort (<b>e</b> and <b>x</b> domains are moved first).</p>
typnotnull	Boolean	Specifies whether the type has a NOT NULL constraint. Currently, it is used for domains only.
typbasetype	oid	If this is a domain (see <b>typtype</b> ), then <b>typbasetype</b> identifies the type that this one is based on. The value is <b>0</b> if this type is not a derived type.
tytypmod	integer	<ul style="list-style-type: none"> <li>• Records the <b>tytypmod</b> to be applied to domains' base types by domains (the value is <b>-1</b> if the base type does not use <b>typmod</b>). This is <b>-1</b> if this type is not a domain.</li> </ul>
typndims	integer	If a field is an array, <b>typndims</b> is the number of array dimensions. This is <b>0</b> for types other than domains over array types.

Name	Type	Description
typcollation	oid	Sorting rule of a specified type. For details about the values, see the <a href="#">PG_COLLATION</a> system catalog. (0 if sequencing is not supported.)
typdefaultbin	pg_node_tree	nodeToString() representation of a default expression for the type if the value is non-null. Currently, this column is only used for domains.
typdefault	text	The value is <b>NULL</b> if a type has no associated default value. If <b>typdefaultbin</b> is not set to <b>NULL</b> , <b>typdefault</b> must contain a default expression represented by <b>typdefaultbin</b> . If <b>typdefaultbin</b> is <b>NULL</b> and <b>typdefault</b> is not, then <b>typdefault</b> is the external representation of the type's default value, which can be fed to the type's input converter to produce a constant.
typacl	aclitem[]	Access permission.
typelemmod	integer	This column has been deprecated.

## 12.2.95 PG\_USER\_MAPPING

**PG\_USER\_MAPPING** records mappings from local users to remote.

This system catalog is accessible only to system administrators. Common users can query the [PG\\_USER\\_MAPPINGS](#) view.

**Table 12-94** PG\_USER\_MAPPING columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
umuser	oid	<a href="#">PG_AUTHID.oid</a>	OID of the local role being mapped (0 if the user mapping is public)
umserver	oid	<a href="#">PG_FOREIGN_SERVER.oid</a>	OID of the foreign server that contains the mapping
umoptions	text[]	-	User mapping specific options, expressed in a string in the format of keyword=value

## 12.2.96 PG\_USER\_STATUS

PG\_USER\_STATUS records the states of users who access the database. This system catalog is accessible only to users with the system administrator permission.

**Table 12-95** PG\_USER\_STATUS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
roid	oid	ID of a role.
failcount	integer	Number of failed attempts.
locktime	timestamp with time zone	By default, the creation date of the role is displayed. If the role is locked by the administrator or the role is locked because the number of login failures exceeds the threshold, the date when the role is locked is displayed.
rolstatus	smallint	Role state. <ul style="list-style-type: none"> <li>• <b>0</b>: normal</li> <li>• <b>1</b>: The role is locked for a specific period of time because the failed login attempts exceed the threshold.</li> <li>• <b>2</b>: The role is locked by the administrator.</li> </ul>
permspace	bigint	Size of the permanent table storage space used by a role.
tempspace	bigint	Size of the temporary table storage space used by a role.
password expired	smallint	Specifies whether a password is valid. <ul style="list-style-type: none"> <li>• <b>0</b>: The password is valid.</li> <li>• <b>1</b>: The password is invalid.</li> </ul>

## 12.2.97 PGXC\_CLASS

PGXC\_CLASS records replicated or distributed information for each table.

**Table 12-96** PGXC\_CLASS columns

Name	Type	Description
pcrelid	oid	OID of the table.
pclocatortype	"char"	Locator type. <ul style="list-style-type: none"> <li>• <b>H</b>: Hash</li> <li>• <b>G</b>: Range</li> <li>• <b>L</b>: List</li> <li>• <b>M</b>: Modulo</li> <li>• <b>N</b>: Round Robin</li> <li>• <b>R</b>: Replication</li> </ul>
pchashalgorithm	smallint	Distributed tuple using the hash algorithm.
pchashbuckets	smallint	Value of a harsh container.
pgroup	name	Node group name.
redistributed	"char"	Indicates that a table has been redistributed.
redis_order	integer	Redistribution sequence. Tables whose values are <b>0</b> will not be redistributed in this round of redistribution.
pcattnum	int2vector	Column number used as a distributed key.
nodeoids	oidvector_extend	List of distributed table node OIDs.
options	text	Extension status information. This is a reserved column in the system.

## 12.2.98 PGXC\_GROUP

PGXC\_GROUP records information about storage node groups.

**Table 12-97** PGXC\_GROUP columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
group_name	name	Name of a node group.

Name	Type	Description
in_redistributi on	"char"	Specifies whether redistribution is required. Valid value: <ul style="list-style-type: none"> <li>• <b>n</b>: The node group is not redistributed.</li> <li>• <b>y</b>: The source node group is in redistribution.</li> <li>• <b>t</b>: The destination node group is in redistribution.</li> </ul>
group_membe rs	oidvector_ext end	Node OID list of the node group.
group_buckets	text	Distributed data bucket group.
is_installation	Boolean	Specifies whether to install a sub-cluster. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
group_acl	aclitem[]	Access permission.
group_kind	"char"	Node group type. The options are as follows: <ul style="list-style-type: none"> <li>• <b>i</b>: installation node group</li> <li>• <b>n</b>: node group in a common cluster</li> <li>• <b>e</b>: elastic cluster</li> </ul>
group_parent	oid	For a child node group, this field indicates the OID of the parent node group. For a parent node group, this field is left blank.

## 12.2.99 PGXC\_NODE

PGXC\_NODE records information about cluster nodes.

### NOTICE

- PGXC\_NODE stores information about database instance nodes. The PGXC\_NODE system catalog has specific meanings only on CNs and its data is valid and correct.
- In the result of querying the PGXC\_NODE system catalog on a DN, only the **node\_id** column is meaningful. Other columns are meaningless and invalid.

**Table 12-98** PGXC\_NODE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).

Name	Type	Description
node_name	name	Node name.
node_type	"char"	Node type. <ul style="list-style-type: none"><li>● <b>C</b>: CN.</li><li>● <b>D</b>: DN.</li><li>● <b>S</b>: standby DN.</li></ul>
node_port	integer	Port ID of the node.
node_host	name	Host name or IP address of a node. (If a virtual IP address is configured, its value is a virtual IP address.)
node_port1	integer	Port number of a replication node.
node_host1	name	Host name or IP address of a replication node. (If a virtual IP address is configured, its value is a virtual IP address.)
hostis_primary	Boolean	Specifies whether a primary/standby switchover occurs on the current node. <ul style="list-style-type: none"><li>● <b>t</b> (true): yes</li><li>● <b>f</b> (false): no</li></ul>
nodeis_primary	Boolean	Specifies whether the current node is preferred to execute non-query operations in the <b>replication</b> table. <ul style="list-style-type: none"><li>● <b>t</b> (true): yes</li><li>● <b>f</b> (false): no</li></ul>
nodeis_preferred	Boolean	Specifies whether the current node is preferred to execute queries in the <b>replication</b> table. <ul style="list-style-type: none"><li>● <b>t</b> (true): yes</li><li>● <b>f</b> (false): no</li></ul>
node_id	integer	Node ID. The value is obtain by calculating the value of <b>node_name</b> using the hash function.
sctp_port	integer	Port used by the TCP proxy communications library of the primary node to listen to the data channel. (The SCTP communications library is no longer supported in the current version.)
control_port	integer	Port used by the TCP proxy communications library of the primary node to listen to the control channel.
sctp_port1	integer	Port used by the TCP proxy communications library of the standby node to listen to the data channel. (The SCTP communications library is no longer supported in the current version.)

Name	Type	Description
control_port1	integer	Port used by the TCP proxy communications library of the standby node to listen to the control channel.
nodeis_central	Boolean	Specifies whether the current node is a central control node. It is used only for CNs and is invalid for DNs. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
nodeis_active	Boolean	Specifies whether the current node is normal. It is used to mark whether the CN is removed and is invalid for DNs. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>

## 12.2.100 PGXC\_REDISTB

**PGXC\_REDISTB** is created during scale-out for each database to record the redistribution status of user tables. It will be deleted after scale-out. Only users with the **connect** permission can view the information.

**Table 12-99**

Name	Type	Description
relname	name	Name of a user table.
nspname	name	Name of the tablespace that contains the table.
pcrelid	oid	OID of a table.
plocator type	character	Locator type. H: hash M: Modulo N: Round Robin R: Replicate
pchashalgorithm	smallint	Distributed tuple using the hash algorithm.
pchashbuckets	smallint	Value of a harsh container.
pgroup	name	Node group to which a table belongs.

Name	Type	Description
redistribut ed	character	Catalog status. <b>i</b> : Table is being redistributed. <b>y</b> : Table redistribution is complete. <b>n</b> : Table has not been redistributed. <b>d</b> : The redistribution is complete, but the temporary table has not been deleted.
redis_orde r	interger	Table redistribution sequence. The default value is <b>1024</b> . The value <b>0</b> indicates that the table is not redistributed. A smaller value indicates that the table is redistributed first.
pcattnum	int2vector	Column number used as a distribution key.
nodeoids	oidvector_e xtend	Node ID of the node group where the table is located.
internal_m ask	integer	Whether reloption contains internal information. Values are as follows: <b>0X0</b> : The internal mask is disabled. <b>0X8000</b> : The internal mask is enabled. <b>0X01</b> : The insert operation is not allowed in this table. <b>0X02</b> : The delete operation is not allowed in this table. <b>0X04</b> : The alter operation is not allowed in this table. <b>0X08</b> : The select operation is not allowed in this table. <b>0X0100</b> : The update operation is not allowed in this table.
table_size	bigint	Size of a table, in bytes.

## 12.2.101 PGXC\_SLICE

PGXC\_SLICE is a system catalog created for recording range distribution and list distribution details. Currently, range interval cannot be used to automatically scale out shards. It is reserved in the system catalog.

**Table 12-100** PGXC\_SLICE columns

Name	Type	Description
relname	name	Table name or shard name, which is distinguished by <b>type</b> .

Name	Type	Description
type	"char"	<ul style="list-style-type: none"> <li>'t': <b>relname</b> is the table name.</li> <li>'s': <b>relname</b> is the shard name.</li> </ul>
strategy	"char"	<ul style="list-style-type: none"> <li>'r': range distributed table.</li> <li>'l': list distributed table.</li> </ul> This value will be extended for subsequent interval shards.
relid	oid	OID of the distributed table to which the tuple belongs.
referenc eoid	oid	OID of the referenced distributed table, which is used for slice reference table creation syntax.
sindex	integer	Position of the current boundary in a shard when the table is a list distributed table.
interval	text[]	Reserved column.
transitb oundary	text[]	Reserved column.
transitn o	integer	Reserved column.
nodeoid	oid	When <b>relname</b> is set to a shard name, <b>nodeoid</b> indicates the OID of the DN where the shard data is stored.
boundar ies	text[]	When <b>relname</b> is set to a shard name, this parameter indicates the boundary value of the shard.
specifie d	boolean	Check whether the DN corresponding to the current segment is explicitly specified in the DDL.
sliceord er	integer	User-defined shard sequence.

## 12.2.102 PLAN\_TABLE\_DATA

**PLAN\_TABLE\_DATA** stores plan information collected by **EXPLAIN PLAN**. Different from the **PLAN\_TABLE** view, the system catalog **PLAN\_TABLE\_DATA** stores **EXPLAIN PLAN** information collected by all sessions and users.

**Table 12-101** PLAN\_TABLE\_DATA columns

Name	Type	Description
session_id	text	Session that inserts the data. Its value consists of a service thread start timestamp and a service thread ID. Values are constrained by <b>NOT NULL</b> .

Name	Type	Description
user_id	oid	User who inserts the data. Values are constrained by <b>NOT NULL</b> .
statement_id	character varying(30)	Query tag specified by a user
plan_id	bigint	Query plan ID. The ID is automatically generated in the plan generation phase and is used by kernel engineers for debugging.
id	integer	Node ID in a plan
operation	character varying(30)	Operation description
options	character varying(255)	Operation action
object_name	name	Name of an operated object. It is defined by users.
object_type	character varying(30)	Object type
object_owner	name	Schema to which an object belongs. It is defined by users.
projection	character varying(4000)	Returned column information
cost	double precision	Execution cost estimated by the optimizer for an operator
cardinality	double precision	Number of rows estimated by the optimizer for an operator

 **NOTE**

- **PLAN\_TABLE\_DATA** records data of all users and sessions on the current node. Only administrators can access all the data. Common users can view their own data in the **PLAN\_TABLE** view.
- Data of inactive (exited) sessions is cleaned from **PLAN\_TABLE\_DATA** by **gs\_clean** after being stored in this system catalog for a certain period of time (5 minutes by default). You can also manually run **gs\_clean -C** to delete inactive session data from the table.
- Data is automatically inserted into **PLAN\_TABLE\_DATA** after **EXPLAIN PLAN** is executed. Therefore, do not manually insert data into or update data in **PLAN\_TABLE\_DATA**. Otherwise, data in **PLAN\_TABLE\_DATA** may be disordered. When you need to delete data from a table, it is recommended that you use the **PLAN\_TABLE** view.
- Information in the **statement\_id**, **object\_name**, **object\_owner**, and **projection** columns is stored in letter cases specified by users and information in other columns is stored in uppercase.

## 12.2.103 STATEMENT\_HISTORY

Displays information about statements executed on the current node. To query this system catalog, you must have the SYSADMIN permission. The result can be queried only in the system database but cannot be queried in the user database.

The constraints on the query of this system catalog are as follows:

- Data must be queried in the Postgres database. No data exists in other databases.
- This system catalog is controlled by **track\_stmt\_stat\_level**. The default value is **OFF,L0**, where the first part controls full SQL statements, and the second part controls slow SQL statements. For details about the record level of each field, see the following table. To ensure system performance, you are advised to use the SET statement to change the value of this parameter so that the parameter takes effect only for the current session.
- For slow SQL statements, if the value of **track\_stmt\_stat\_level** is not **OFF** and the SQL execution time exceeds the value of **log\_min\_duration\_statement**, the SQL statement is recorded as a slow SQL statement.

### NOTE

In the current version, the keyword FOR UPDATE cannot be identified or normalized. For example, **SELECT \* FROM table;** and **SELECT \* FROM table FOR UPDATE WAIT N;** are normalized into the same normalized SQL statement and displayed in the **query** column. SQL statements containing the keyword FOR UPDATE can be distinguished by the **query\_plan** column. The execution plan contains **lockRows**.

**Table 12-102** STATEMENT\_HISTORY columns

Name	Type	Description	Record Level
db_name	name	Database name.	L0
schema_name	name	Schema name.	L0
origin_node	integer	Node name.	L0
user_name	name	Username.	L0
application_name	text	Name of the application that sends a request.	L0
client_addr	text	IP address of the client that sends a request.	L0
client_port	integer	Port number of the client that sends a request.	L0
unique_query_id	bigint	ID of the normalized SQL statement.	L0

Name	Type	Description	Record Level
debug_query_id	bigint	ID of the unique SQL statement. Some statements are not unique. For example, the value of <b>debug_query_id</b> in the Parse packet, DCL statements, and TCL statements is <b>0</b> .	L0
query	text	Normalized SQL (available only on CNs). When <b>track_stmt_parameter</b> is enabled, complete SQL statements are displayed.	L0
start_time	timestamp with time zone	Time when a statement starts.	L0
finish_time	timestamp with time zone	Time when a statement ends.	L0
slow_sql_threshold	bigint	Standard for slow SQL statement execution.	L0
transaction_id	bigint	Transaction ID.	L0
thread_id	bigint	ID of an execution thread.	L0
session_id	bigint	Session ID of a user.	L0
n_soft_parse	bigint	Number of soft parsing times. The value of <b>n_soft_parse</b> plus the value of <b>n_hard_parse</b> may be greater than the value of <b>n_calls</b> because the number of subqueries are not counted in the value of <b>n_calls</b> .	L0
n_hard_parse	bigint	Number of hard parsing times. The value of <b>n_soft_parse</b> plus the value of <b>n_hard_parse</b> may be greater than the value of <b>n_calls</b> because the number of subqueries are not counted in the value of <b>n_calls</b> .	L0
query_plan	text	Statement execution plan. For <b>query_plan</b> of Slow SQL, if <b>execution_time</b> is greater than <b>slow_sql_threshold</b> , <b>query_plan</b> may be empty in scenarios such as deadlock.	L0
n_returned_rows	bigint	Number of rows in the result set returned by the <b>SELECT</b> statement.	L0

Name	Type	Description	Record Level
n_tuples_fetched	bigint	Number of rows randomly scanned.	L0
n_tuples_returned	bigint	Number of rows sequentially scanned.	L0
n_tuples_inserted	bigint	Number of rows inserted.	L0
n_tuples_updated	bigint	Number of rows updated.	L0
n_tuples_deleted	bigint	Number of rows deleted.	L0
n_blocks_fetched	bigint	Number of buffer block access times.	L0
n_blocks_hit	bigint	Number of buffer block hits.	L0
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: $\mu$ s).	L0
cpu_time	bigint	CPU time (unit: $\mu$ s).	L0
execution_time	bigint	Execution time in the executor (unit: $\mu$ s).	L0
parse_time	bigint	SQL parsing time (unit: $\mu$ s).	L0
plan_time	bigint	SQL plan generation time (unit: $\mu$ s).	L0
rewrite_time	bigint	SQL rewriting time (unit: $\mu$ s).	L0
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: $\mu$ s).	L0
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: $\mu$ s).	L0
data_io_time	bigint	I/O time (unit: $\mu$ s).	L0

Name	Type	Description	Record Level
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, CNs communicate with each other, CNs communicate with the client, and CNs communicate with DN through physical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <b>{"time":xxx, "n_calls":xxx, "size":xxx}</b> .	L0
net_rcv_info	text	Network status of messages received through a physical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, CNs communicate with each other, CNs communicate with the client, and CNs communicate with DN through physical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <b>{"time":xxx, "n_calls":xxx, "size":xxx}</b> .	L0
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, DNs of different shards communicate with each other through logical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <b>{"time":xxx, "n_calls":xxx, "size":xxx}</b> .	L0

Name	Type	Description	Record Level
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, DNs of different shards communicate with each other through logical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <code>{"time":xxx, "n_calls":xxx, "size":xxx}</code> .	L0
lock_count	bigint	Number of locks.	L0
lock_time	bigint	Time required for locking.	L1
lock_wait_count	bigint	Number of lock waits.	L0
lock_wait_time	bigint	Time required for lock waiting.	L1
lock_max_count	bigint	Maximum number of locks.	L0
lwlock_count	bigint	Number of lightweight locks (reserved).	L0
lwlock_wait_count	bigint	Number of lightweight lock waits.	L0
lwlock_time	bigint	Time required for lightweight locking (reserved).	L1
lwlock_wait_time	bigint	Time required for lightweight lock waiting.	L1

Name	Type	Description	Record Level
details	bytea	<p>List of wait events and statement lock events.</p> <p>When the value of <b>track_stmt_stat_level</b> is greater than or equal to <b>L0</b>, the list of wait events starts to be recorded. It displays statistics about wait events on the current node. For details about key events in the kernel, see <a href="#">Table 12-331</a>, <a href="#">Table 12-332</a>, <a href="#">Table 12-333</a>, and <a href="#">Table 12-334</a>. You can also view the list of all events in the system in the <code>wait_event_info</code> view. For details about the impact of each transaction lock on services, see <a href="#">LOCK</a>.</p> <p>When the value of <b>track_stmt_stat_level</b> is <b>L2</b>, the list of statement lock events is recorded. The list records events in chronological order. The number of records is affected by the value of the <b>track_stmt_details_size</b> parameter.</p> <p>This field is in binary format and needs to be read by using the parsing function <code>pg_catalog.statement_detail_decode</code>. For details, see <a href="#">Table 7-94</a>.</p> <p>Events include:</p> <ul style="list-style-type: none"> <li>● Start locking.</li> <li>● Complete locking.</li> <li>● Start lock waiting.</li> <li>● Complete lock waiting.</li> <li>● Start unlocking.</li> <li>● Complete unlocking.</li> <li>● Start lightweight lock waiting.</li> <li>● Complete lightweight lock waiting.</li> </ul>	L0/L2
is_slow_sql	Boolean	<p>Specifies whether the SQL statement is a slow SQL statement.</p> <ul style="list-style-type: none"> <li>● <b>t</b> (true): yes.</li> <li>● <b>f</b> (false): no</li> </ul>	L0
trace_id	text	Driver-specific trace ID, which is associated with an application request.	L0

Name	Type	Description	Record Level
advise	text	Risk information that may cause the SQL statement to be a slow SQL statement.	L0
parent_unique_sql_id	bigint	Normalized SQL ID of the outer SQL statement. For statements executed in a stored procedure, the value is the normalized SQL ID of the statement that calls the stored procedure. For statements outside the stored procedure, the value is <b>0</b> .	L0

## 12.2.104 STREAMING\_STREAM

**STREAMING\_STREAM** records the metadata of all STREAM objects.

**Table 12-103** STREAMING\_STREAM column

Name	Type	Description
relid	oid	STREAM object ID.
queries	bytea	Bitmap mapping of the CONTVIEW corresponding to the STREAM.

## 12.2.105 STREAMING\_CONT\_QUERY

**STREAMING\_CONT\_QUERY** records the metadata of all CONTVIEW objects.

**Table 12-104** STREAMING\_CONT\_QUERY columns

Name	Type	Description
id	integer	Unique identifier of the CONTVIEW object.
type	"char"	CONTVIEW type. <ul style="list-style-type: none"> <li>'r' indicates that the CONTVIEW is based on the row-store model.</li> </ul>
relid	oid	CONTVIEW object ID.
defrelid	oid	ID of the continuous computing rule view corresponding to CONTVIEW.

Name	Type	Description
active	Boolean	Specifies whether the CONTVIEW is in the continuous computing state. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes</li><li>• <b>f</b> (false): no</li></ul>
streamrelid	oid	ID of STREAM corresponding to CONTVIEW.
matrelid	oid	ID of the materialized table corresponding to CONTVIEW.
lookupidxid	oid	ID of GROUP LOOK UP INDEX corresponding to CONTVIEW. This column is for internal use and is available only in row-store tables.
step_factor	smallint	CONTVIEW step mode. The main values are <b>0</b> (no overlapping window) and <b>1</b> (sliding window, with one step).
ttn	integer	Value of <b>ttn_interval</b> set by CONTVIEW.
ttn_attno	smallint	Number of a time column corresponding to the TTL function set by CONTVIEW.
dictrelid	oid	ID of the dictionary table corresponding to CONTVIEW.
grpnum	smallint	Number of dimension columns in the CONTVIEW continuous computing rule. This column is for internal use.
grpidx	int2vector	Index of the dimension column in TARGET LIST in the CONTVIEW continuous computing rule. This column is for internal use.

## 12.3 System Views

### 12.3.1 ADM\_ARGUMENTS

ADM\_ARGUMENTS displays parameter information of all stored procedures or functions. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both the PG\_CATALOG and SYS schemas.

**Table 12-105** ADM\_ARGUMENTS columns

Name	Type	Description
owner	character varying(128)	Owner of a function or stored procedure.

Name	Type	Description
object_name	character varying(128)	Name of a function or stored procedure.
object_id	oid	OID of a function or stored procedure.
overload	character varying(40)	<i>n</i> th overloaded function of the name.
subprogram_id	numeric	Not supported. Its value is <b>NULL</b> .
argument_name	character varying(128)	Parameter name.
position	numeric	Position of the parameter in the parameter list. The value is <b>0</b> for the return value of the function by default.
sequence	numeric	Sequence of a parameter, which starts from 1, with the return type before all parameters.
data_level	numeric	Nesting depth of parameters of the composite type. The value is fixed at <b>0</b> because only one line is displayed for each parameter.
data_type	character varying(64)	Data type of a parameter.
defaulted	character varying(1)	Specifies whether a parameter has a default value. <ul style="list-style-type: none"> <li>• <b>Y</b>: yes.</li> <li>• <b>N</b>: no.</li> </ul>
default_value	text	Not supported. Its value is <b>NULL</b> .
default_length	numeric	Not supported. Its value is <b>NULL</b> .
in_out	character varying(9)	Input and output attributes of a parameter: <ul style="list-style-type: none"> <li>• <b>IN</b>: input parameter.</li> <li>• <b>OUT</b>: output parameter.</li> <li>• <b>IN_OUT</b>: input and output parameters.</li> <li>• <b>VARIADIC</b>: VARIADIC parameter.</li> </ul>
data_length	numeric	Not supported. Its value is <b>NULL</b> .
data_precision	numeric	Not supported. Its value is <b>NULL</b> .
data_scale	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
radix	numeric	Radix of a number, which is <b>10</b> when the data type is smallint, integer, bigint, numeric, or float, and is <b>NULL</b> for other data types.
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
type_owner	character varying(128)	Owner of a data type.
type_name	character varying(128)	Parameter type name.
type_subname	character varying(128)	Not supported. Its value is <b>NULL</b> .
type_link	character varying(128)	Not supported. Its value is <b>NULL</b> .
type_object_type	character varying(7)	Types of types of the <b>type_owner</b> , <b>type_name</b> , and <b>type_subname</b> columns: <ul style="list-style-type: none"><li>• <b>TABLE</b>: The parameter is of the table type.</li><li>• <b>VIEW</b>: The parameter is of the view type.</li><li>• Set this column to <b>NULL</b> for other types.</li></ul>
pls_type	character varying(128)	Name of the PL/SQL type for parameters of the number type. Otherwise, this column is empty.
char_length	numeric	Not supported. Its value is <b>NULL</b> .
char_used	character varying(1)	Not supported. Set it to <b>B</b> if the data type is varchar, nvarchar2, bpchar, or char, and to <b>NULL</b> for other data types.
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .

### 12.3.2 ADM\_AUDIT\_OBJECT

ADM\_AUDIT\_OBJECT displays the audit trail records of all objects in the database. This view exists in both PG\_CATALOG and SYS schemas. By default, only the system administrator can access this view. Common users can access the view only after being authorized. The **action\_name** column of GaussDB is different from that of the ORA database in terms of the audit actions. The type of the

**transactionid** column is the same as that in the ORA database. In GaussDB, the **sql\_text** column is the parsed SQL statement, which is not completely the same as the executed SQL statement.

**Table 12-106** ADM\_AUDIT\_OBJECT columns

Name	Type	Description
os_username	character varying(255)	Not supported. Its value is <b>NULL</b> .
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. Its value is <b>NULL</b> .
terminal	character varying(255)	Not supported. Its value is <b>NULL</b> .
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created in the local database session time zone (user login date and time of the entry created by the audit session).
owner	character varying(128)	Creator of the object affected by the operation.
obj_name	character varying(128)	Name of the object affected by the operation.
action_name	character varying(28)	Action name corresponding to the numeric code in the <b>ACTION</b> column in DBA_AUDIT_TRAIL
new_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
new_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
ses_actions	character varying(19)	Not supported. Its value is <b>NULL</b> .
comment_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .
sessionid	numeric	Not supported. Its value is <b>NULL</b> .
entryid	numeric	Not supported. Its value is <b>NULL</b> .
statementid	numeric	Not supported. Its value is <b>NULL</b> .
returncode	numeric	Not supported. Its value is <b>NULL</b> .
priv_used	character varying(40)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
client_id	character varying(128)	Not supported. Its value is <b>NULL</b> .
econtext_id	character varying(64)	Not supported. Its value is <b>NULL</b> .
session_cpu	numeric	Not supported. Its value is <b>NULL</b> .
extended_timestamp	timestamp(6) with time zone	Timestamp when an audit trail entry is created (user login timestamp of the entry created by the audit session in UTC).
proxy_sessionid	numeric	Not supported. Its value is <b>NULL</b> .
global_uid	character varying(32)	Not supported. Its value is <b>NULL</b> .
instance_numeric	numeric	Not supported. Its value is <b>NULL</b> .
os_process	character varying(16)	Not supported. Its value is <b>NULL</b> .
transactionid	text	Identifier of the transaction that accesses or modifies an object.
scn	numeric	Not supported. Its value is <b>NULL</b> .
sql_bind	Ncharacter varying(2000)	Not supported. Its value is <b>NULL</b> .
sql_text	Ncharacter varying(2000)	SQL text of the query.
obj_edition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .

### 12.3.3 ADM\_AUDIT\_SESSION

ADM\_AUDIT\_SESSION displays all audit trail records concerning CONNECT and DISCONNECT. The audit information of GaussDB is mainly obtained through the pg\_query\_audit function. This view exists in both PG\_CATALOG and SYS schemas. Only users with the AUDITADMIN attribute can view audit information. The **action\_name** column of GaussDB is inconsistent with the audit action of database ORA, and the **transactionid** column is consistent with the type of **transactionid** data in database ORA.

**Table 12-107** ADM\_AUDIT\_SESSION columns

Name	Type	Description
os_username	character varying(255)	Not supported. Its value is <b>NULL</b> .
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. Its value is <b>NULL</b> .
terminal	character varying(128)	Not supported. Its value is <b>NULL</b> .
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created (user login date and time of the entry created by the audit session)
action_name	character varying(28)	Action name corresponding to the numeric code in the <b>ACTION</b> column in DBA_AUDIT_TRAIL.
logoff_time	timestamp(0) without time zone	Not supported. Its value is <b>NULL</b> .
logoff_lread	numeric	Not supported. Its value is <b>NULL</b> .
logoff_pread	numeric	Not supported. Its value is <b>NULL</b> .
logoff_lwrite	numeric	Not supported. Its value is <b>NULL</b> .
logoff_dlock	character varying(40)	Not supported. Its value is <b>NULL</b> .
sessionid	numeric	Not supported. Its value is <b>NULL</b> .
returncode	numeric	Not supported. Its value is <b>NULL</b> .
client_id	character varying(128)	Not supported. Its value is <b>NULL</b> .
session_cpu	numeric	Not supported. Its value is <b>NULL</b> .
extended_timestamp	timestamp(6) with time zone	Time zone of the timestamp when an audit trail entry is created (user login date and time of the entry created by the audit session in UTC)
proxy_sessionid	numeric	Not supported. Its value is <b>NULL</b> .
global_uid	character varying(32)	Not supported. Its value is <b>NULL</b> .
instance_numeric	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
os_process	character varying(16)	Not supported. Its value is <b>NULL</b> .

### 12.3.4 ADM\_AUDIT\_STATEMENT

ADM\_AUDIT\_STATEMENT displays all grant and revoke audit trail entries. The audit information of GaussDB is mainly obtained through the pg\_query\_audit function. This view exists in both PG\_CATALOG and SYS schemas. Only users with the AUDITADMIN attribute can view audit information. The **action\_name** column of GaussDB is different from that of the ORA database in terms of the audit actions. The type of the **transactionid** column is the same as that in the ORA database. In GaussDB, the **sql\_text** column is the parsed SQL statement, which is not completely the same as the executed SQL statement.

**Table 12-108** ADM\_AUDIT\_STATEMENT columns

Name	Type	Description
os_username	character varying(255)	Not supported. Its value is <b>NULL</b> .
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. Its value is <b>NULL</b> .
terminal	character varying(255)	Not supported. Its value is <b>NULL</b> .
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created in the local database session time zone (user login date and time of the entry created by the audit session).
owner	character varying(128)	Creator of the object affected by the operation.
obj_name	character varying(128)	Name of the object affected by the operation.
action	numeric	Not supported. Its value is <b>NULL</b> .
action_name	character varying(28)	Action type corresponding to the code in the action column.
new_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
new_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
obj_privilege	character varying(32)	Not supported. Its value is <b>NULL</b> .
sys_privilege	character varying(40)	Not supported. Its value is <b>NULL</b> .
admin_option	character varying(1)	Not supported. Its value is <b>NULL</b> .
grantee	character varying(128)	Not supported. Its value is <b>NULL</b> .
audit_option	character varying(40)	Not supported. Its value is <b>NULL</b> .
ses_actions	character varying(19)	Not supported. Its value is <b>NULL</b> .
logoff_time	timestamp(0) without time zone	Not supported. Its value is <b>NULL</b> .
logoff_lread	numeric	Not supported. Its value is <b>NULL</b> .
logoff_pread	numeric	Not supported. Its value is <b>NULL</b> .
logoff_lwrite	numeric	Not supported. Its value is <b>NULL</b> .
logoff_dlock	character varying(40)	Not supported. Its value is <b>NULL</b> .
comment_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .
sessionid	numeric	Not supported. Its value is <b>NULL</b> .
entryid	numeric	Not supported. Its value is <b>NULL</b> .
statementid	numeric	Not supported. Its value is <b>NULL</b> .
returncode	numeric	Not supported. Its value is <b>NULL</b> .
priv_used	character varying(40)	Not supported. Its value is <b>NULL</b> .
client_id	character varying(128)	Not supported. Its value is <b>NULL</b> .
econtext_id	character varying(64)	Not supported. Its value is <b>NULL</b> .
session_cpu	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
extended_timestamp	timestamp(6) with time zone	Time zone of the timestamp when an audit trail entry is created (user login date and time of the entry created by the audit session in UTC).
proxy_sessionid	numeric	Not supported. Its value is <b>NULL</b> .
global_uid	character varying(32)	Not supported. Its value is <b>NULL</b> .
instance_number	numeric	Not supported. Its value is <b>NULL</b> .
os_process	character varying(16)	Not supported. Its value is <b>NULL</b> .
transactionid	text	Identifier of the transaction that accesses or modifies an object.
scn	numeric	Not supported. Its value is <b>NULL</b> .
sql_bind	nvarchar2(2000)	Not supported. Its value is <b>NULL</b> .
sql_text	nvarchar2	SQL text of the query.
obj_edition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
dbid	numeric	Not supported. Its value is <b>NULL</b> .
rls_info	clob	Not supported. Its value is <b>NULL</b> .
current_user	character varying(128)	Not supported. Its value is <b>NULL</b> .

### 12.3.5 ADM\_AUDIT\_TRAIL

ADM\_AUDIT\_TRAIL displays all standard audit trail entries. The audit information of GaussDB is mainly obtained through the `pg_query_audit` function. This view exists in both `PG_CATALOG` and `SYS` schemas. Only users with the `AUDITADMIN` attribute can view audit information. The **action\_name** column of GaussDB is different from that of the ORA database in terms of the audit actions. The type of the **transactionid** column is the same as that in the ORA database. In GaussDB, the **sql\_text** column is the parsed SQL statement, which is not completely the same as the executed SQL statement.

---

**⚠ CAUTION**

In a distributed system, the `pg_query_audit` function can only be used to query the audit information of the current node.

---

**Table 12-109** ADM\_AUDIT\_TRAIL columns

Name	Type	Description
os_username	character varying(255)	Not supported. Its value is <b>NULL</b> .
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. Its value is <b>NULL</b> .
terminal	character varying(255)	Not supported. Its value is <b>NULL</b> .
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created in the local database session time zone (user login date and time of the entry created by the audit session).
owner	character varying(128)	Creator of the object affected by the operation.
obj_name	character varying(128)	Name of the object affected by the operation.
action	numeric	Not supported. Its value is <b>NULL</b> .
action_name	character varying(28)	Action type corresponding to the code in the action column.
new_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
new_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
obj_privilege	character varying(32)	Not supported. Its value is <b>NULL</b> .
sys_privilege	character varying(40)	Not supported. Its value is <b>NULL</b> .
admin_option	character varying(1)	Not supported. Its value is <b>NULL</b> .
grantee	character varying(128)	Not supported. Its value is <b>NULL</b> .
audit_option	character varying(40)	Not supported. Its value is <b>NULL</b> .
ses_actions	character varying(19)	Not supported. Its value is <b>NULL</b> .
logoff_time	timestamp(0) without time zone	Not supported. Its value is <b>NULL</b> .
logoff_lread	numeric	Not supported. Its value is <b>NULL</b> .
logoff_pread	numeric	Not supported. Its value is <b>NULL</b> .
logoff_lwrite	numeric	Not supported. Its value is <b>NULL</b> .
logoff_dlock	character varying(40)	Not supported. Its value is <b>NULL</b> .
comment_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
sessionid	numeric	Not supported. Its value is <b>NULL</b> .
entryid	numeric	Not supported. Its value is <b>NULL</b> .
statementid	numeric	Not supported. Its value is <b>NULL</b> .
returncode	numeric	Not supported. Its value is <b>NULL</b> .
priv_used	character varying(40)	Not supported. Its value is <b>NULL</b> .
client_id	character varying(128)	Not supported. Its value is <b>NULL</b> .
econtext_id	character varying(64)	Not supported. Its value is <b>NULL</b> .
session_cpu	numeric	Not supported. Its value is <b>NULL</b> .
extended_timestamp	timestamp(6) with time zone	Time zone of the timestamp when an audit trail entry is created (user login date and time of the entry created by the audit session in UTC).
proxy_session_id	numeric	Not supported. Its value is <b>NULL</b> .
global_uid	character varying(32)	Not supported. Its value is <b>NULL</b> .
instance_number	numeric	Not supported. Its value is <b>NULL</b> .
os_process	character varying(16)	Not supported. Its value is <b>NULL</b> .
transactionid	text	Identifier of the transaction that accesses or modifies an object.
scn	numeric	Not supported. Its value is <b>NULL</b> .
sql_bind	nvarchar2(2000)	Not supported. Its value is <b>NULL</b> .
sql_text	nvarchar2	SQL text of the query.
obj_edition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
dbid	numeric	Not supported. Its value is <b>NULL</b> .
rls_info	clob	Not supported. Its value is <b>NULL</b> .
current_user	character varying(128)	Not supported. Its value is <b>NULL</b> .

### 12.3.6 ADM\_COL\_COMMENTS

ADM\_COL\_COMMENTS displays information about table column comments in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-110** ADM\_COL\_COMMENTS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.
comments	text	Comments.
origin_con_id	numeric	Not supported. Its value is <b>0</b> .
schema	character varying(64)	Name of the namespace to which the column belongs.

## 12.3.7 ADM\_COL\_PRIVS

ADM\_COL\_PRIVS displays permission granting information about all columns. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-111** ADM\_COL\_PRIVS columns

Name	Type	Description
grantor	character varying(128)	Name of the user who grants the permission.
owner	character varying(128)	Object owner.
grantee	character varying(128)	Name of the user or role to which the permission is granted.
table_schema	character varying(128)	Schema of an object.
table_name	character varying(128)	Object name.
column_name	character varying(128)	Column name.
privilege	character varying(40)	Permission on a column.
grantable	character varying(3)	Specifies whether to grant privileges. <ul style="list-style-type: none"><li>• <b>YES</b>: The privileges are granted.</li><li>• <b>NO</b>: The privileges are not granted.</li></ul>

Name	Type	Description
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

## 12.3.8 ADM\_COLL\_TYPES

ADM\_COLL\_TYPES displays information about all collection types. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-112** ADM\_COLL\_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of a cluster.
type_name	character varying(128)	Name of a collection.
coll_type	character varying(128)	Description of a set.
upper_bound	numeric	Not supported. Its value is <b>NULL</b> .
elem_type_mod	character varying(7)	Not supported. Its value is <b>NULL</b> .
elem_type_name	character varying(128)	Name of the data type or user-defined type on which the collection is based.
length	numeric	Not supported. Its value is <b>NULL</b> .
precision	numeric	Not supported. Its value is <b>NULL</b> .
scale	numeric	Not supported. Its value is <b>NULL</b> .
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
elem_storage	character varying(7)	Not supported. Its value is <b>NULL</b> .
nulls_stored	character varying(3)	Not supported. Its value is <b>NULL</b> .
char_used	character varying(1)	Not supported. Its value is <b>NULL</b> .

## 12.3.9 ADM\_CONS\_COLUMNS

ADM\_CONS\_COLUMNS displays information about constraint columns in database tables. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-113** ADM\_CONS\_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
constraint_name	character varying(64)	Constraint name.
table_name	character varying(64)	Name of a constraint-related table.
column_name	character varying(64)	Name of a constraint-related column.
position	smallint	Position of a column in a table.

## 12.3.10 ADM\_CONSTRAINTS

ADM\_CONSTRAINTS displays information about table constraints in database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-114** ADM\_CONSTRAINTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
constraint_name	character varying(64)	Constraint name.
constraint_type	text	Constraint type. <ul style="list-style-type: none"><li>● <b>c</b>: check constraint.</li><li>● <b>f</b>: foreign key constraint.</li><li>● <b>p</b>: primary key constraint.</li><li>● <b>u</b>: unique constraint.</li></ul>
table_name	character varying(64)	Name of a constraint-related table.

Name	Type	Description
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint).
index_name	character varying(64)	Name of the constraint-related index (only for the unique constraint and primary key constraint).
status	character varying(8)	Constraint status.
generated	character varying(14)	Not supported. Its value is <b>NULL</b> .
search_condition	text	Not supported. Its value is <b>NULL</b> .
search_condition_v c	character varying(4000)	Not supported. Its value is <b>NULL</b> .
r_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
r_constraint_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
delete_rule	character varying(9)	Not supported. Its value is <b>NULL</b> .
con_deferrable	character varying(14)	Not supported. Its value is <b>NULL</b> .
deferred	character varying(9)	Not supported. Its value is <b>NULL</b> .
validated	character varying(13)	Not supported. Its value is <b>NULL</b> .
bad	character varying(3)	Not supported. Its value is <b>NULL</b> .
rely	character varying(4)	Not supported. Its value is <b>NULL</b> .
last_change	date	Not supported. Its value is <b>NULL</b> .
invalid	character varying(7)	Not supported. Its value is <b>NULL</b> .
view_related	character varying(14)	Not supported. Its value is <b>NULL</b> .
origin_con_id	character varying(256)	Not supported. Its value is <b>NULL</b> .

### 12.3.11 ADM\_DATA\_FILES

ADM\_DATA\_FILES displays the description of database files. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-115** ADM\_DATA\_FILES columns

Name	Type	Description
tablespace_name	name	Name of the tablespace to which a file belongs.
bytes	double precision	Length of the file in bytes.

## 12.3.12 ADM\_DEPENDENCIES

ADM\_DEPENDENCIES displays the dependency relationships between types, tables, views, stored procedures, functions, and triggers in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-116** ADM\_DEPENDENCIES columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	character varying(18)	Object type.
referenced_owner	name	Owner of the referenced object.
referenced_name	name	Name of the referenced object.
referenced_type	character varying(18)	Type of the referenced object.
referenced_link_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
dependency_type	character varying(4)	Not supported. Its value is <b>NULL</b> .

## 12.3.13 ADM\_DIRECTORIES

ADM\_DIRECTORIES displays all directory objects in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-117** ADM\_DIRECTORIES columns

Name	Type	Description
owner	oid	Directory owner.
directory_name	name	Directory name.
directory_path	text	OS path name of a directory.
origin_con_id	character varying(256)	Not supported. Its value is <b>NULL</b> .

## 12.3.14 ADM\_HIST\_SNAPSHOT

ADM\_HIST\_SNAPSHOT records the WDR snapshot data stored in the current system. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas. This view can be accessed only when the GUC parameter **enable\_wdr\_snapshot** is set to **on**. To access the PG\_CATALOG.ADM\_HIST\_SNAPSHOT and SYS.ADM\_HIST\_SNAPSHOT views, you must have the permission to access the **snapshot schema**, **snapshot**, and **tables\_snap\_timestamp** tables.

**Table 12-118** ADM\_HIST\_SNAPSHOT columns

Name	Type	Description
snap_id	bigint	Unique snapshot ID.
dbid	oid	Database ID of the snapshot.
instance_number	oid	Not supported. The value is the same as that of DBID.
startup_time	timestamp(3) without time zone	Instance start time.
begin_interval_time	timestamp without time zone	Start time of the snapshot interval (the end time of the last snapshot).
end_interval_time	timestamp without time zone	End time of the snapshot interval, that is, the actual time when the snapshot is taken (the end time of this snapshot).
flush_elapsed	interval	Amount of time to perform the snapshot.
snap_level	numeric	Not supported. Its value is <b>NULL</b> .
error_count	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
snap_flag	numeric	Not supported. Its value is <b>NULL</b> .
snap_timezone	interval day to second(0)	Time offset between the snapshot time zone and the coordinated universal time (UTC).
begin_interval_time_tz	timestamp with time zone	Start time of the snapshot interval with the time zone (end time of the last snapshot).
end_interval_time_tz	timestamp with time zone	End time of the snapshot interval, that is, the actual time when the snapshot is taken (the end time of this snapshot), with the time zone.
con_id	numeric	Not supported. Its value is <b>0</b> .

### 12.3.15 ADM\_HIST\_SQL\_PLAN

ADM\_HIST\_SQL\_PLAN displays plan information collected by the current user by running the EXPLAIN PLAN statement. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-119** ADM\_HIST\_SQL\_PLAN columns

Name	Type	Description
dbid	text	Database ID.
sql_id	character varying(30)	Session that inserts the data. Its value consists of a service thread start timestamp and a service thread ID. Values are constrained by <b>NOT NULL</b> .
id	integer	Number assigned to each step in the execution plan.
plan_hash_value	bigint	Query ID.
operation	character varying(30)	Operation description.
options	character varying(255)	Operation option.
object_name	name	Name of an operated object. It is defined by users.
object_node	character varying(128)	Not supported. Its value is <b>NULL</b> .
object#	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
object_owner	name	Object number of a table or an index.
object_alias	character varying(261)	Not supported. Its value is <b>NULL</b> .
object_type	character varying(30)	Object type.
optimizer	character varying(20)	Not supported. Its value is <b>NULL</b> .
parent_id	numeric	Not supported. Its value is <b>NULL</b> .
depth	numeric	Not supported. Its value is <b>NULL</b> .
position	numeric	Not supported. Its value is <b>NULL</b> .
search_columns	numeric	Not supported. Its value is <b>NULL</b> .
cost	double precision	Execution cost estimated by the optimizer for an operator.
cardinality	double precision	Cardinality estimated by the optimizer for an operator to access table records.
bytes	numeric	Not supported. Its value is <b>NULL</b> .
other_tag	character varying(35)	Not supported. Its value is <b>NULL</b> .
partition_start	character varying(64)	Not supported. Its value is <b>NULL</b> .
partition_stop	character varying(64)	Not supported. Its value is <b>NULL</b> .
partition_id	numeric	Not supported. Its value is <b>NULL</b> .
other	character varying(4000)	Not supported. Its value is <b>NULL</b> .
distribution	character varying(20)	Not supported. Its value is <b>NULL</b> .
cpu_cost	numeric	Not supported. Its value is <b>NULL</b> .
io_cost	numeric	Not supported. Its value is <b>NULL</b> .
temp_space	numeric	Not supported. Its value is <b>NULL</b> .
access_predicates	character varying(4000)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
filter_predicates	character varying(4000)	Not supported. Its value is <b>NULL</b> .
projection	character varying(4000)	Returned column information.
time	numeric	Not supported. Its value is <b>NULL</b> .
qblock_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
remarks	character varying(4000)	Not supported. Its value is <b>NULL</b> .
timestamp	date	Not supported. Its value is <b>NULL</b> .
other_xml	clob	Not supported. Its value is <b>NULL</b> .
con_dbid	text	Database ID of a container, which is currently set to a value the same as that of <b>dbid</b> .
con_id	numeric	Container ID. Containers are not supported. Its value is <b>0</b> .

## 12.3.16 ADM\_HIST\_SQLSTAT

ADM\_HIST\_SQLSTAT displays information about statements executed on the current node. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

After the WDR snapshot function is enabled (that is, the GUC parameter **enable\_wdr\_snapshot** is set to **on**), users can view the data in this view.

**Table 12-120** ADM\_HIST\_SQLSTAT columns

Name	Type	Description
instance_number	integer	Instance ID of a snapshot.
sql_id	bigint	Query ID.
plan_hash_value	integer	ID of the normalized SQL statement.
module	integer	Name of the module that is executing when the SQL statement is first parsed.

Name	Type	Description
elapsed_time_delta	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: $\mu$ s).
cpu_time_delta	bigint	Time consumed on the CPU (unit: $\mu$ s).
executions_delta	bigint	Increment in the number of executions that have occurred on this object since it was brought into the cache.
iowait_delta	bigint	I/O time (unit: $\mu$ s).
apwait_delta	integer	Delta value of the application wait time.
rows_processed_delta	bigint	Number of rows in the result set returned by the SELECT statement.
snap_id	bigint	Unique snapshot ID.
parsing_schema_name	character varying	Not supported. Its value is <b>NULL</b> .
disk_reads_delta	bigint	Not supported. Its value is <b>NULL</b> .
buffer_reads_delta	bigint	Not supported. Its value is <b>NULL</b> .
clwait_delta	bigint	Not supported. Its value is <b>NULL</b> .

### 12.3.17 ADM\_HIST\_SQLTEXT

ADM\_HIST\_SQLTEXT displays information about statements executed on the current node. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

After the WDR snapshot function is enabled (that is, the GUC parameter **enable\_wdr\_snapshot** is set to **on**), users can view the data in this view.

**Table 12-121** ADM\_HIST\_SQLTEXT columns

Name	Type	Description
dbid	integer	Database ID.
sql_id	bigint	Query ID.
sql_text	clob	Text corresponding to the query.
command_type	integer	Not supported. Its value is <b>0</b> .

Name	Type	Description
con_dbid	integer	Database ID of a container, which is currently set to a value the same as that of <b>dbid</b> .
con_id	integer	Container ID. Containers are not supported. Its value is <b>0</b> .

## 12.3.18 ADM\_IND\_COLUMNS

ADM\_IND\_COLUMNS displays information about index columns in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-122** ADM\_IND\_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	name	Column name.
column_position	smallint	Position of the column in the index.
column_length	numeric	Length of the column. For the variable-length type, its value is <b>NULL</b> .
char_length	numeric	Maximum length of a column, in bytes.
descend	character varying	Specifies whether columns are sorted in descending ( <b>DESC</b> ) or ascending ( <b>ASC</b> ) order.
collated_column_id	numeric	Not supported. Its value is <b>NULL</b> .

## 12.3.19 ADM\_IND\_EXPRESSIONS

ADM\_IND\_EXPRESSIONS displays information about expression indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-123** ADM\_IND\_EXPRESSIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
column_expression	text	Function-based index expression of a specified column.
column_position	smallint	Position of the column in the index.

## 12.3.20 ADM\_IND\_PARTITIONS

ADM\_IND\_PARTITIONS displays the partition information about local indexes of level-1 partitioned table in the database. Each local index partition of level-1 index partitioned table in the database (if any) has a row of records in ADM\_IND\_PARTITIONS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-124** ADM\_IND\_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index to which an index partition belongs.
index_name	character varying(64)	Name of the partitioned table index to which the index partition belongs.
partition_name	character varying(64)	Name of the index partition.
def_tablespace_name	name	Tablespace name of the index partition.

Name	Type	Description
high_value	text	Upper limit of the partition corresponding to the index partition. <ul style="list-style-type: none"><li>• For range partitioning and interval partitioning, the upper limit of each partition is displayed.</li><li>• For list partitioning, the value list of each partition is displayed.</li><li>• For hash partitioning, the number of each partition is displayed.</li></ul>
index_partition_usable	Boolean	Specifies whether the index partition is available. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes.</li><li>• <b>f</b> (false): no.</li></ul>
schema	character varying(64)	Schema of the partitioned table index to which the index partition belongs.
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition.
composite	character varying(3)	Specifies whether the index is a local index on the level-2 partitioned table. This table does not store level-2 partition information. Therefore, the value is <b>NO</b> .
subpartition_count	numeric	Number of level-2 partitions in a partition. This table does not store level-2 partition information. Therefore, the value is <b>0</b> .
partition_position	numeric	Position of an index partition in the index.
status	character varying(8)	Specifies whether the index partition is available.
tablespace_name	name	Name of the tablespace that contains the partition.
pct_free	numeric	Percentage of minimum available space in a block.

Name	Type	Description
ini_trans	numeric	Initial number of transactions. The default value is <b>4</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is <b>128</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Specifies whether changes to an index are logged.
compression	character varying(13)	Specifies whether an index compression is enabled for a partitioned index.
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
clustering_factor	numeric	Sequence of a row in the table based on the value of the index. You need to run the <b>analyze</b> command to collect statistics.
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Actual buffer pool of a partition.
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
pct_direct_access	numeric	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table. The distributed system does not support interval partitioning. Therefore, the value of this parameter is <b>NO</b> .
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
orphaned_entries	character varying(3)	Not supported. Its value is <b>NULL</b> .

## 12.3.21 ADM\_IND\_SUBPARTITIONS

ADM\_IND\_SUBPARTITIONS displays the partition information about local indexes of level-2 partitioned table in the database (excluding global indexes on partitioned tables). Each local index partition of level-2 index partitioned table in the database (if any) has a row of records in ADM\_IND\_SUBPARTITIONS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas. The distributed system does not support level-2 partitions. Currently, all columns in this table are set to **NULL**.

**Table 12-125** ADM\_IND\_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
index_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
high_value	text	Not supported. Its value is <b>NULL</b> .
high_value_length	numeric	Not supported. Its value is <b>NULL</b> .
partition_position	numeric	Not supported. Its value is <b>NULL</b> .
subpartition_position	numeric	Not supported. Its value is <b>NULL</b> .
status	character varying(8)	Not supported. Its value is <b>NULL</b> .
tablespace_name	character varying(30)	Not supported. Its value is <b>NULL</b> .
pct_free	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Not supported. Its value is <b>NULL</b> .
max_trans	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Not supported. Its value is <b>NULL</b> .
compression	character varying(13)	Not supported. Its value is <b>NULL</b> .
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
clustering_factor	numeric	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .
buffer_pool	character varying(7)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Not supported. Its value is <b>NULL</b> .
segment_created	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .

## 12.3.22 ADM\_INDEXES

ADM\_INDEXES displays all indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-126** ADM\_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_name	character varying(64)	Name of the table corresponding to the index.
uniqueness	text	Specifies whether the index is a unique index. <ul style="list-style-type: none"><li>• <b>UNIQUE</b>: unique index.</li><li>• <b>NONUNIQUE</b>: non-unique index.</li></ul>

Name	Type	Description
partitioned	character(3)	<p>Specifies whether the index has the property of partitioned tables.</p> <ul style="list-style-type: none"> <li>• <b>Yes:</b> The index has the property of a partitioned table.</li> <li>• <b>No:</b> The index does not have the property of a partitioned table.</li> </ul>
generated	character varying(1)	<p>Specifies whether the index name is generated by the system.</p> <ul style="list-style-type: none"> <li>• <b>y:</b> The index name is generated by the system.</li> <li>• <b>n:</b> The index name is not generated by the system.</li> </ul>
index_type	character varying(27)	<p>Index type.</p> <ul style="list-style-type: none"> <li>• <b>NORMAL:</b> Index attributes are simple references, and the expression tree is empty.</li> <li>• <b>FUNCTION-BASED NORMAL:</b> Expression trees are used for index attributes that are not simple column references.</li> </ul>
table_owner	character varying(128)	Owner of an index object.
table_type	character(11)	<p>Type of the index object.</p> <ul style="list-style-type: none"> <li>• <b>TABLE:</b> The index object is of the table type.</li> </ul>
tablespace_name	character varying(30)	Name of a tablespace that contains the index.

Name	Type	Description
status	character varying(8)	Status of a non-partitioned index. <ul style="list-style-type: none"> <li>• <b>VALID</b>: Non-partitioned indexes can be used for query.</li> <li>• <b>UNUSABLE</b>: The non-partitioned index is unavailable.</li> <li>• <b>N/A</b>: The index has the property of a partitioned table.</li> </ul>
compression	character varying(13)	Not supported. Its value is <b>NULL</b> .
prefix_length	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Not supported. Its value is <b>NULL</b> .
max_trans	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
pct_threshold	numeric	Not supported. Its value is <b>NULL</b> .
include_column	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
pct_free	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
logging	character varying(3)	Not supported. Its value is <b>NULL</b> .
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
clustering_factor	numeric	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .
degree	character varying(40)	Not supported. Its value is <b>NULL</b> .
instances	character varying(40)	Not supported. Its value is <b>NULL</b> .
temporary	character varying(1)	Not supported. Its value is <b>NULL</b> .
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
buffer_pool	character varying(7)	Not supported. Its value is <b>NULL</b> .
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
duration	character varying(15)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
pct_direct_access	numeric	Not supported. Its value is <b>NULL</b> .
ityp_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
ityp_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_status	character varying(12)	Not supported. Its value is <b>NULL</b> .
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
funcidx_status	character varying(8)	Not supported. Its value is <b>NULL</b> .
join_index	character varying(3)	Not supported. Its value is <b>NULL</b> .
iot_redundant_pkey_elim	character varying(3)	Not supported. Its value is <b>NULL</b> .
dropped	character varying(3)	Not supported. Its value is <b>NULL</b> .
visibility	character varying(9)	Not supported. Its value is <b>NULL</b> .
domidx_management	character varying(14)	Not supported. Its value is <b>NULL</b> .
segment_created	character varying(3)	Not supported. Its value is <b>NULL</b> .
orphaned_entries	character varying(3)	Not supported. Its value is <b>NULL</b> .
indexing	character varying(7)	Not supported. Its value is <b>NULL</b> .
auto	character varying(3)	Not supported. Its value is <b>NULL</b> .

## 12.3.23 ADM\_OBJECTS

ADM\_OBJECTS displays all database objects in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-127** ADM\_OBJECTS columns

Name	Type	Description
owner	name	Object owner.
object_name	name	Object name.
object_id	oid	Object OID.
object_type	name	Object type. For example, table, schema, and index.
namespace	oid	Namespace containing the object.
temporary	character(1)	Specifies whether the object is a temporary object.
status	character varying(7)	Object status. <ul style="list-style-type: none"><li>• <b>valid</b></li><li>• <b>invalid</b></li></ul>
subobject_name	name	Subobject name of the object.
generated	character(1)	Specifies whether the object name is generated by the system.
created	timestamp with time zone	Time when the object was created.
last_ddl_time	timestamp with time zone	Last time when the object was modified.
default_collation	character varying(100)	Default collation of objects.
data_object_id	numeric	Not supported. Its value is <b>NULL</b> .
timestamp	character varying(19)	Not supported. Its value is <b>NULL</b> .
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
edition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
sharing	character varying(18)	Not supported. Its value is <b>NULL</b> .
editionable	character varying(1)	Not supported. Its value is <b>NULL</b> .
oracle_maintained	character varying(1)	Not supported. Its value is <b>NULL</b> .
application	character varying(1)	Not supported. Its value is <b>NULL</b> .
duplicated	character varying(1)	Not supported. Its value is <b>NULL</b> .
sharded	character varying(1)	Not supported. Its value is <b>NULL</b> .
created_appid	numeric	Not supported. Its value is <b>NULL</b> .
modified_appid	numeric	Not supported. Its value is <b>NULL</b> .
created_vsnid	numeric	Not supported. Its value is <b>NULL</b> .
modified_vsnid	numeric	Not supported. Its value is <b>NULL</b> .

**NOTICE**

For details about the value ranges of **created** and **last\_ddl\_time**, see [PG\\_OBJECT](#).

## 12.3.24 ADM\_PART\_COL\_STATISTICS

ADM\_PART\_COL\_STATISTICS displays the column statistics and histogram information about all table partitions in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-128** ADM\_PART\_COL\_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Owner of the partitioned table.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Table partition name.
column_name	character varying(4000)	Column name.

Name	Type	Description
num_distinct	numeric	Not supported. Its value is <b>NULL</b> .
low_value	raw	Not supported. Its value is <b>NULL</b> .
high_value	raw	Not supported. Its value is <b>NULL</b> .
density	numeric	Not supported. Its value is <b>NULL</b> .
num_nulls	numeric	Not supported. Its value is <b>NULL</b> .
num_buckets	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
notes	character varying(63)	Not supported. Its value is <b>NULL</b> .
avg_col_len	numeric	Not supported. Its value is <b>NULL</b> .
histogram	character varying(15)	Not supported. Its value is <b>NULL</b> .
schema	character varying(64)	Name of the namespace to which the column belongs.

### 12.3.25 ADM\_PART\_INDEXES

**ADM\_PART\_INDEXES** displays information about all partitioned table indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-129** ADM\_PART\_INDEXES columns

Name	Type	Description
def_tablespace_name	name	Tablespace name of the partitioned table index
index_owner	character varying(64)	Name of the owner of a partitioned table index
index_name	character varying(64)	Name of the partitioned table index
partition_count	bigint	Number of index partitions of the partitioned table index
partitioning_key_count	integer	Number of partition keys of the partitioned table
partitioning_type	text	Partition policy of the partitioned table <b>NOTE</b> For details about the supported partitioning policies of the current partitioned table, see <a href="#">CREATE TABLE PARTITION</a> .
schema	character varying(64)	Name of the schema to which the partitioned table index belongs
table_name	character varying(64)	Name of the partitioned table to which the partitioned table index belongs

## 12.3.26 ADM\_PART\_TABLES

ADM\_PART\_TABLES displays information about all partitioned tables in the database. By default, only system administrators can access this view. Common users can access the view only after being authorized. This view exists in both the PG\_CATALOG and SYS schemas.

**Table 12-130** ADM\_PART\_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table.
table_name	character varying(64)	Name of the partitioned table.

Name	Type	Description
partitioning_type	text	Partitioning policy of the partitioned table. <b>NOTE</b> For details about the supported partitioning policies of the current partitioned table, see <a href="#">CREATE TABLE PARTITION</a> .
partition_count	bigint	Number of partitions of the partitioned table.
partitioning_key_count	integer	Number of partition keys of the partitioned table.
def_tablespace_name	name	Tablespace name of the partitioned table.
schema	character varying(64)	Schema of the partitioned table.
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, <b>NONE</b> is displayed. The distributed system does not support level-2 partitions. Set this parameter to <b>NONE</b> .
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is <b>1</b> for a level-2 partitioned table and <b>0</b> for a level-1 partitioned table. The distributed system does not support level-2 partitions. Set this parameter to <b>0</b> .
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table. The distributed system does not support level-2 partitions. Set this parameter to <b>0</b> .
status	character varying(8)	Not supported. Its value is <b>valid</b> .
def_pct_free	numeric	Default value of <b>PCTFREE</b> used when a partition is added.
def_pct_used	numeric	Not supported. Its value is <b>NULL</b> .
def_ini_trans	numeric	Default value of <b>INITRANS</b> used when a partition is added.
def_max_trans	numeric	Default value of <b>MAXTRANS</b> used when a partition is added.

Name	Type	Description
def_initial_extent	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_next_extent	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_min_extents	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_max_extents	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_max_size	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_pct_increase	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_freelists	numeric	Not supported. Its value is <b>NULL</b> .
def_freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
def_logging	character varying(7)	Not supported. Its value is <b>NULL</b> .
def_compression	character varying(8)	Default compression mode used when a partition is added. Value range: <ul style="list-style-type: none"> <li>• <b>NONE</b></li> <li>• <b>ENABLED</b></li> <li>• <b>DISABLED</b></li> </ul>
def_compress_for	character varying(30)	Default compression mode used when a partition is added. <b>NOTE</b> For available compression modes and compression levels, see <a href="#">WITH ( { storage_paramet...</a>
def_buffer_pool	character varying(7)	Not supported. Its value is <b>DEFAULT</b> .
def_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
def_cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
ref_ptn_constraint_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
interval	character varying(1000)	Interval.
autolist	character varying(3)	Not supported. Its value is <b>NO</b> .

Name	Type	Description
interval_subpartition	character varying(1000)	Not supported. Its value is <b>NULL</b> .
autolist_subpartition	character varying(3)	Not supported. Its value is <b>NO</b> .
is_nested	character varying(3)	Not supported. Its value is <b>NO</b> .
def_segment_creation	character varying(4)	Currently, the segment-page mode is not supported. When the segment-page mode is enabled, the value is <b>YES</b> .
def_indexing	character varying(3)	Not supported. Its value is <b>ON</b> .
def_inmemory	character varying(8)	Not supported. Its value is <b>NONE</b> .
def_inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
def_inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
def_inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
def_inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
def_read_only	character varying(3)	Not supported. Its value is <b>NO</b> .
def_cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .

### 12.3.27 ADM\_PROCEDURES

ADM\_PROCEDURES displays information about all stored procedures, functions, and triggers in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-131** ADM\_PROCEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure, function, or trigger.
object_name	character varying(64)	Name of a stored procedure, function, or trigger.
procedure_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
object_id	oid	OID of a stored procedure, function, or trigger.
subprogram_id	numeric	Not supported. Its value is <b>NULL</b> .
overload	character varying(40)	<i>n</i> th overloaded function of the name.
object_type	character varying(13)	Object type name.
aggregate	character varying(3)	Specifies whether the function is an aggregate function: <ul style="list-style-type: none"><li>• <b>YES</b></li><li>• <b>NO</b></li></ul>
pipelined	character varying(3)	Not supported. Its value is <b>NO</b> .
impltypeowner	character varying(128)	Owner of an implementation type.
impltypename	character varying(128)	Name of an implementation type.
parallel	character varying(3)	Not supported. Its value is <b>NO</b> .
interface	character varying(3)	Not supported. Its value is <b>NO</b> .
deterministic	character varying(3)	Not supported. Its value is <b>NO</b> .

Name	Type	Description
authid	character varying(12)	Specifies whether to use the creator permission or caller permission: <ul style="list-style-type: none"><li>• <b>DEFINER</b>: The creator permission is used.</li><li>• <b>CURRENT_USER</b>: The caller permission is used.</li></ul> This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
result_cache	character varying(3)	Not supported. Its value is <b>NULL</b> .
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .
polymorphic	character varying(5)	Not supported. Its value is <b>NULL</b> .
argument_number	smallint	Number of input parameters in a stored procedure.

### 12.3.28 ADM\_ROLE\_PRIVS

ADM\_ROLE\_PRIVS displays information about roles granted to all users and roles. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-132** ADM\_ROLE\_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
granted_role	character varying(128)	Role to be granted.
admin_option	character varying(3)	Specifies whether the grant contains the <b>ADMIN</b> option. <ul style="list-style-type: none"><li>• <b>YES</b></li><li>• <b>NO</b></li></ul>
delegate_option	character varying(3)	Not supported. Its value is <b>NULL</b> .
default_role	character varying(3)	Not supported. Its value is <b>NULL</b> .
os_granted	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.29 ADM\_ROLES

ADM\_ROLES displays information about database roles. This view is accessible only to system administrators. This view exists in both PG\_CATALOG and SYS schemas.

Table 12-133 ADM\_ROLES columns

Name	Type	Description
role	character varying(128)	Role name.
role_id	oid	Role ID.
authentication_type	text	Role authentication mechanism. <ul style="list-style-type: none"> <li>• <b>password</b>: Password authentication is required.</li> <li>• <b>null</b>: Authentication is not required.</li> </ul>
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
oracle_maintained	character varying(1)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .
implicit	character varying(3)	Not supported. Its value is <b>NULL</b> .
external_name	character varying(4000)	Not supported. Its value is <b>NULL</b> .

### 12.3.30 ADM\_SCHEDULER\_JOB\_ARGS

ADM\_SCHEDULER\_JOB\_ARG displays parameters related to all jobs in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-134** ADM\_SCHEDULER\_JOB\_ARGS columns

Name	Type	Description
owner	character varying(128)	Owner of the job to which the parameter belongs.
job_name	character varying(128)	Name of the job to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter, which can be a user-defined data type.
value	character varying(4000)	Parameter value.
anydata_value	character varying(4000)	Not supported. Its value is <b>NULL</b> .
out_argument	character varying(5)	Reserved column. Its value is <b>NULL</b> .

### 12.3.31 ADM\_SCHEDULER\_JOBS

ADM\_SCHEDULER\_JOBS displays information about all DBE\_SCHEDULER scheduled jobs in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-135** ADM\_SCHEDULER\_JOBS columns

Name	Type	Description
owner	name	Owner of a scheduled job.
job_name	text	Name of a scheduled job.
job_subname	character varying(128)	Not supported. Its value is <b>NULL</b> .
job_style	text	Style of a scheduled job, which is specified during creation. Its value can only be " <b>REGULAR</b> ". If this parameter is not specified, the value is <b>NULL</b> .
job_creator	name	Creator of a scheduled job.

Name	Type	Description
client_id	character varying(65)	Not supported. Its value is <b>NULL</b> .
global_uid	character varying(33)	Not supported. Its value is <b>NULL</b> .
program_owner	character varying(4000)	Owner of a program referenced by a scheduled job.
program_name	text	Name of the program referenced by a scheduled job.
job_type	character varying(16)	Inline program type of a scheduled job. The options are as follows: <ul style="list-style-type: none"> <li>• <b>PLSQL_BLOCK</b>: anonymous block of a stored procedure.</li> <li>• <b>STORED_PROCEDURE</b>: stored procedure that is saved.</li> <li>• <b>EXTERNAL_SCRIPT</b>: external script.</li> </ul>
job_action	text	Program content of a scheduled job.
number_of_arguments	text	Number of parameters of a scheduled job.
schedule_owner	character varying(4000)	Not supported. Its value is <b>NULL</b> .
schedule_name	text	Name of the schedule referenced by a scheduled job.
schedule_type	character varying(12)	Not supported. Its value is <b>NULL</b> .
start_date	timestamp without time zone	Start time of a scheduled job.
repeat_interval	text	Period of a scheduled job.
event_queue_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
event_queue_name	character varying(128)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
event_queue_agent	character varying(523)	Not supported. Its value is <b>NULL</b> .
event_condition	character varying(4000)	Not supported. Its value is <b>NULL</b> .
event_rule	character varying(261)	Not supported. Its value is <b>NULL</b> .
file_watcher_owner	character varying(261)	Not supported. Its value is <b>NULL</b> .
file_watcher_name	character varying(261)	Not supported. Its value is <b>NULL</b> .
end_date	timestamp without time zone	End time of a scheduled job.
job_class	text	Name of the scheduled job class to which a scheduled job belongs.
enabled	Boolean	Status of a scheduled job.
auto_drop	text	Status of the automatic deletion function of a scheduled job.
restart_on_recovery	character varying(5)	Not supported. Its value is <b>NULL</b> .
restart_on_failure	character varying(5)	Not supported. Its value is <b>NULL</b> .
state	"char"	Status of a scheduled job.
job_priority	numeric	Not supported. Its value is <b>NULL</b> .
run_count	numeric	Not supported. Its value is <b>NULL</b> .
uptime_run_count	numeric	Not supported. Its value is <b>NULL</b> .
max_runs	numeric	Not supported. Its value is <b>NULL</b> .
failure_count	smallint	Number of scheduled job failures.
uptime_failure_count	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
max_failures	numeric	Maximum number of failures allowed before the status of a scheduled job is marked as broken.
retry_count	numeric	Not supported. Its value is <b>NULL</b> .
last_start_date	timestamp without time zone	Last time when a scheduled job was started.
last_run_duration	interval day to second(6)	Last execution duration of a scheduled job.
next_run_date	timestamp without time zone	Next execution time of a scheduled job.
schedule_limit	interval day to second(0)	Not supported. Its value is <b>NULL</b> .
max_run_duration	interval day to second(0)	Not supported. Its value is <b>NULL</b> .
logging_level	character varying(11)	Not supported. Its value is <b>NULL</b> .
store_output	character varying(5)	Specifies whether to store the output information of all scheduled jobs.
stop_on_window_close	character varying(5)	Not supported. Its value is <b>NULL</b> .
instance_stickiness	character varying(5)	Not supported. Its value is <b>NULL</b> .
raise_events	character varying(4000)	Not supported. Its value is <b>NULL</b> .
system	character varying(5)	Not supported. Its value is <b>NULL</b> .
job_weight	numeric	Not supported. Its value is <b>NULL</b> .
nls_env	character varying(4000)	Not supported. Its value is <b>NULL</b> .
source	character varying(128)	Not supported. Its value is <b>NULL</b> .
number_of_destinations	numeric	Not supported. Its value is <b>NULL</b> .
destination_owner	character varying(261)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
destination	text	Target name of a scheduled job.
credential_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
credential_name	text	Certificate name of a scheduled job.
instance_id	oid	OID of the current database
deferred_drop	character varying(5)	Not supported. Its value is <b>NULL</b> .
allow_runs_in_restricted_mode	character varying(5)	Not supported. Its value is <b>NULL</b> .
comments	text	Comments of a scheduled job.
flags	numeric	Not supported. Its value is <b>NULL</b> .
restartable	character varying(5)	Not supported. Its value is <b>NULL</b> .
has_constraints	character varying(5)	Not supported. Its value is <b>NULL</b> .
connect_credential_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
connect_credential_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
fail_on_script_error	character varying(5)	Not supported. Its value is <b>NULL</b> .

### 12.3.32 ADM\_SCHEDULER\_PROGRAM\_ARGS

ADM\_SCHEDULER\_PROGRAM\_ARG displays parameters related to all programs in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-136** ADM\_SCHEDULER\_PROGRAM\_ARGS columns

Name	Type	Description
owner	character varying(128)	Owner of the program to which the parameter belongs.

Name	Type	Description
program_name	character varying(128)	Name of the program to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter, which can be a user-defined data type.
metadata_attribute	character varying(19)	Not supported. Its value is <b>NULL</b> .
default_value	character varying(4000)	Default parameter value.
default_anydata_value	character varying(4000)	Not supported. Its value is <b>NULL</b> .
out_argument	character varying(5)	Reserved column. Its value is <b>NULL</b> .

### 12.3.33 ADM\_SCHEDULER\_PROGRAMS

ADM\_SCHEDULER\_PROGRAMS displays information about all programs that can be scheduled in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-137** ADM\_SCHEDULER\_PROGRAMS columns

Name	Type	Description
owner	name	Owner of a scheduler program
program_name	text	Name of a scheduler program

Name	Type	Description
program_type	character varying(16)	Type of the scheduler. The options are as follows: <ul style="list-style-type: none"> <li>• <b>PLSQL_BLOCK</b>: anonymous block of a stored procedure.</li> <li>• <b>STORED_PROCEDURE</b>: stored procedure that is saved.</li> <li>• <b>EXTERNAL_SCRIPT</b>: external script.</li> </ul>
program_action	text	Action performed by a scheduler program
number_of_arguments	numeric	Number of scheduler program parameters
enabled	character varying(5)	Specifies whether a scheduler program is enabled.
comments	text	Comments of a scheduler program
detached	character varying(5)	Not supported. Its value is <b>NULL</b> .
schedule_limit	interval day to second(0)	Not supported. Its value is <b>NULL</b> .
priority	numeric	Not supported. Its value is <b>NULL</b> .
weight	numeric	Not supported. Its value is <b>NULL</b> .
max_runs	numeric	Not supported. Its value is <b>NULL</b> .
max_failures	numeric	Not supported. Its value is <b>NULL</b> .
max_run_duration	interval day to second(0)	Not supported. Its value is <b>NULL</b> .
has_constraints	character varying(5)	Not supported. Its value is <b>NULL</b> .
nls_env	character varying(4000)	Not supported. Its value is <b>NULL</b> .

## 12.3.34 ADM\_SCHEDULER\_RUNNING\_JOBS

ADM\_SCHEDULER\_RUNNING\_JOBS displays information about all DBE\_SCHEDULER jobs that are being executed in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-138** ADM\_SCHEDULER\_RUNNING\_JOBS columns

Name	Type	Description
owner	character varying(128)	Owner of a scheduled job.
job_name	character varying(128)	Name of a scheduled job.
job_subname	character varying(128)	Not supported. Its value is <b>NULL</b> .
job_style	character varying(17)	Style of a scheduled job, which is specified during creation. Its value can only be " <b>REGULAR</b> ". If this parameter is not specified, the value is <b>NULL</b> .
detached	character varying(5)	Not supported. Its value is <b>NULL</b> .
session_id	numeric	ID of the session that executes a scheduled job.
slave_process_id	numeric	Not supported. Its value is <b>NULL</b> .
slave_os_process_id	character varying(12)	ID of the process that executes a scheduled job.
running_instance	numeric	Not supported. Its value is <b>NULL</b> .
resource_consumer_group	character varying(32)	Not supported. Its value is <b>NULL</b> .
elapsed_time	interval day to second(2)	Execution duration of a scheduled job.
cpu_used	interval day to second(2)	Not supported. Its value is <b>NULL</b> .
destination_owner	character varying(261)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
destination	character varying(261)	Target name of a scheduled job.
credential_name	character varying(128)	Credential name of a scheduled job.
credential_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
log_id	numeric	Not supported. Its value is <b>NULL</b> .

### 12.3.35 ADM\_SEGMENTS

ADM\_SEGMENTS displays the storage space allocated to all segments in the database. It exists in both the PG\_CATALOG and SYS schemas. Only the system administrator can access this view. The information cannot be obtained from the system catalog. The view is empty.

**Table 12-139** ADM\_SEGMENTS columns

Name	Type	Description
owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
segment_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
segment_type	character varying(18)	Not supported. Its value is <b>NULL</b> .
segment_subtype	character varying(10)	Not supported. Its value is <b>NULL</b> .
tablespace_name	character varying(30)	Not supported. Its value is <b>NULL</b> .
header_file	numeric	Not supported. Its value is <b>NULL</b> .
header_block	numeric	Not supported. Its value is <b>NULL</b> .
bytes	numeric	Not supported. Its value is <b>NULL</b> .
blocks	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
extents	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
retention	character varying(7)	Not supported. Its value is <b>NULL</b> .
minretention	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
relative_fno	numeric	Not supported. Its value is <b>NULL</b> .
buffer_pool	character varying(7)	Not supported. Its value is <b>NULL</b> .
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
inmemory	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .

### 12.3.36 ADM\_SEQUENCES

ADM\_SEQUENCES displays information about all sequences in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-140** ADM\_SEQUENCES columns

Name	Type	Description
sequence_owner	character varying(64)	Owner of a sequence.
sequence_name	character varying(64)	Name of the sequence.
min_value	int16	Minimum value of the sequence.
max_value	int16	Maximum value of the sequence.
increment_by	int16	Increment of the sequence.
last_number	int16	Value of the previous sequence.
cache_size	int16	Size of the sequence disk cache.
cycle_flag	character(1)	Specifies whether the sequence is a cyclic sequence. The value can be <b>Y</b> or <b>N</b> . <ul style="list-style-type: none"><li>• <b>Y</b>: It is a cyclic sequence.</li><li>• <b>N</b>: It is not a cyclic sequence.</li></ul>

### 12.3.37 ADM\_SOURCE

ADM\_SOURCE displays the definition information about all stored procedures, functions, and triggers in the database. By default, only the system administrator

can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-141** ADM\_SOURCE columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	name	Object type. The value can be <b>function, procedure, or trigger</b> .
line	numeric	Number of the line in the definition information.
text	text	Text source of the storage object.
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .

### 12.3.38 ADM\_SUBPART\_KEY\_COLUMNS

ADM\_SUBPART\_KEY\_COLUMNS displays information about the partition key columns of level-2 partitioned tables or partitioned indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas. The distributed system does not support level-2 partitioned tables. All columns in this view are set to **NULL**.

**Table 12-142** ADM\_SUBPART\_KEY\_COLUMNS columns

Name	Type	Description
owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
name	character varying(128)	Not supported. Its value is <b>NULL</b> .
object_type	character varying(128)	Not supported. Its value is <b>NULL</b> .
column_name	character varying(4000)	Not supported. Its value is <b>NULL</b> .
column_position	numeric	Not supported. Its value is <b>NULL</b> .
collated_column_id	numeric	Not supported. Its value is <b>NULL</b> .

### 12.3.39 ADM\_SYNONYMS

ADM\_SYNONYMS displays all synonyms in the database. This view is accessible only to system administrators. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-143** ADM\_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym.
schema_name	text	Name of the schema to which the synonym belongs.
synonym_name	text	Synonym name.
table_owner	text	Owner of the associated object. Although the column is called <b>table_owner</b> , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_name	text	Name of the associated object. Although the column is called <b>table_name</b> , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_schema_name	text	Schema name of the associated object. Although the column is called <b>table_schema_name</b> , the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

### 12.3.40 ADM\_SYS\_PRIVS

ADM\_SYS\_PRIVS displays information about system permissions granted to users and roles. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-144** ADM\_SYS\_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
privilege	character varying(40)	System permission System permissions include rolsuper, rolinherit, rolcreatorole, rolcreatedb, rolcatupdate, rolcanlogin, rolreplication, rolauditadmin, rolsystemadmin, roluseft, rolmonitoradmin, roloperatoradmin, and rolpolicyadmin.
admin_option	character varying(3)	Specifies whether the grant contains the <b>ADMIN</b> option. <ul style="list-style-type: none"> <li>• <b>YES</b></li> <li>• <b>NO</b></li> </ul>
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.41 ADM\_TAB\_COLS

ADM\_TAB\_COLS displays information about table and view columns. Each column of each table and view in the database has a row of data in ADM\_TAB\_COLS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas. The number of rows displayed in this view is the same as that in the ADM\_TAB\_COLUMNS view. Only columns are different.

**Table 12-145** ADM\_TAB\_COLS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view.
table_name	character varying(128)	Name of the table or view.
column_name	character varying(128)	Column name.
data_type	character varying(128)	Data type of a column, which can be a user-defined data type.
data_type_mod	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
data_type_owner	character varying(128)	Owner of the data type of a column.
data_length	numeric	Length of the column, in bytes.
data_precision	numeric	Precision of a data type. This column is valid for the numeric data type and <b>NULL</b> for other data types.
data_scale	numeric	Number of decimal places. This column is valid for the numeric data type and <b>0</b> for other data types.
nullable	character varying(1)	Specifies whether a column can be empty ( <b>n</b> for a primary key constraint or NOT NULL constraint).
column_id	numeric	Sequence number of the column when the table is created.
default_length	numeric	Length of the default value of a column, in bytes.
data_default	text	Default value of a column.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Minimum value in a column.
high_value	raw	Maximum value in a column.
density	numeric	Column density.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	date	Last analysis date.
sample_size	numeric	Sample size used to analyze a column.
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
char_col_decl_length	numeric	Declaration length of a column of the character type.

Name	Type	Description
global_stats	character varying(3)	Not supported. Its value is <b>NO</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NO</b> .
avg_col_len	numeric	Average length of a column, in bytes.
char_length	numeric	Column length (in characters) which is valid only for the varchar, nvarchar2, bpchar, and char types.
char_used	character varying(1)	Not supported. Set it to <b>B</b> if the data type is varchar, nvarchar2, bpchar, or char, and to <b>NULL</b> for other data types.
v80_fmt_image	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_upgraded	character varying(3)	Not supported. Its value is <b>YES</b> .
hidden_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
virtual_column	character varying	Specifies whether a column is a virtual column (a generated column). <b>YES</b> <b>NO</b>
segment_column_id	numeric	Not supported. Its value is <b>NULL</b> .
internal_column_id	numeric	Internal sequence number of a column. The value is the same as that of <b>COLUMN_ID</b> .
histogram	character varying(15)	Histogram type: <ul style="list-style-type: none"> <li>• <b>NONE</b>: no histogram.</li> <li>• <b>FREQUENCY</b>: frequency histogram.</li> <li>• <b>EQUI_WIDTH</b>: equal-width histogram.</li> </ul>
qualified_col_name	character varying(64)	Qualified column name, which is the same as <b>COLUMN_NAME</b> .

Name	Type	Description
user_generated	character varying(3)	Not supported. Its value is <b>YES</b> .
default_on_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
identity_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
evaluation_edition	character varying(128)	Not supported. Its value is <b>NULL</b> .
unusable_before	character varying(128)	Not supported. Its value is <b>NULL</b> .
unusable_beginning	character varying(128)	Not supported. Its value is <b>NULL</b> .
collation	character varying(100)	Collation rule of a column. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
collated_column_id	numeric	Not supported. Its value is <b>NULL</b> .
schema	character varying(64)	Name of the namespace to which the column belongs.

## 12.3.42 ADM\_TAB\_COL\_STATISTICS

ADM\_TAB\_COL\_STATISTICS displays column statistics and histogram information extracted from ADM\_TAB\_COLUMNS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-146** ADM\_TAB\_COL\_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.

Name	Type	Description
num_distinct	numeric	Number of different values in a column.
low_value	raw	Low value in a column.
high_value	raw	High value in a column.
density	numeric	<ul style="list-style-type: none"> <li>If there is a histogram on <b>COLUMN_NAME</b>, this column displays the selectivity of values in the histogram that span less than two endpoints. It does not represent the selectivity of values that span two or more endpoints.</li> <li>If no histogram is available on <b>COLUMN_NAME</b>, the value of this column is <b>1/NUM_DISTINCT</b>.</li> </ul>
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
sample_size	numeric	Sample size used to analyze a column.
last_analyzed	timestamp(0) without time zone	Date when a column was last analyzed. After the database is restarted, data loss will occur.
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
notes	character varying(99)	Not supported. Its value is <b>NULL</b> .
avg_col_len	numeric	Average length of a column, in bytes.
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram if it exists. <ul style="list-style-type: none"> <li><b>NONE</b>: no histogram.</li> <li><b>FREQUENCY</b>: frequency histogram.</li> <li><b>EQUI-WIDTH</b>: equal-width histogram.</li> </ul>
scope	character varying(7)	For statistics collected on any table other than global temporary tables, the value is <b>SHARED</b> (indicating that the statistics are shared among all sessions).
schema	character varying(64)	Name of the namespace to which the column belongs.

## 12.3.43 ADM\_TAB\_COLUMNS

ADM\_TAB\_COLUMNS displays information about the columns of tables and views. Each column of each table and view in the database has a row of data in ADM\_TAB\_COLUMNS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-147** ADM\_TAB\_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	character varying(64)	Column name.
data_type	character varying(128)	Data type of the column.
data_type_mod	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_type_owner	character varying(128)	Owner of the data type of a column.
data_length	integer	Length of the column, in bytes.
data_precision	integer	Precision of a data type. This column is valid for the numeric data type and <b>NULL</b> for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and <b>0</b> for other data types.
nullable	bpchar	Specifies whether a column can be empty ( <b>n</b> for a primary key constraint or NOT NULL constraint). <ul style="list-style-type: none"><li>• <b>y</b>: yes.</li><li>• <b>n</b>: no.</li></ul>
column_id	integer	Sequence number of the column when the table is created.
default_length	numeric	Length of the default value of a column, in bytes. This column is left blank if no default value left.
data_default	text	Default value of a column.
num_distinct	numeric	Number of different values in a column.

Name	Type	Description
low_value	raw	Minimum value in a column.
high_value	raw	Maximum value in a column.
density	numeric	Column density.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	date	Last analysis date.
sample_size	numeric	Sample size used to analyze a column.
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
char_col_decl_length	numeric	Declaration length of a column of the character type.
global_stats	character varying(3)	Not supported. Its value is <b>NO</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NO</b> .
avg_col_len	numeric	Average length of a column, in bytes.
char_length	numeric	Column length (in characters) which is valid only for the varchar, nvarchar2, bpchar, and char types.
char_used	character varying(1)	Not supported. Set it to <b>B</b> if the data type is varchar, nvarchar2, bpchar, or char, and to <b>NULL</b> for other data types.
v80_fmt_image	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_upgraded	character varying(3)	Not supported. Its value is <b>YES</b> .
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram if it exists. <ul style="list-style-type: none"> <li>• <b>NONE</b>: no histogram.</li> <li>• <b>FREQUENCY</b>: frequency histogram.</li> <li>• <b>EQUI_WIDTH</b>: equal-width histogram.</li> </ul>
default_on_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
identity_column	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
evaluation_edition	character varying(128)	Not supported. Its value is <b>NULL</b> .
unusable_before	character varying(128)	Not supported. Its value is <b>NULL</b> .
unusable_beginning	character varying(128)	Not supported. Its value is <b>NULL</b> .
collation	character varying(100)	Collation rule of a column. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
comments	text	Comment of a column.
schema	character varying(64)	Name of the namespace to which the column belongs.

## 12.3.44 ADM\_TAB\_COMMENTS

ADM\_TAB\_COMMENTS displays comments about all tables and views in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-148** ADM\_TAB\_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view.
table_name	character varying(64)	Name of the table or view.
comments	text	Comments.
schema	character varying(64)	Name of the namespace to which the table belongs.

## 12.3.45 ADM\_TAB\_HISTOGRAMS

ADM\_TAB\_HISTOGRAMS displays the histogram information about all tables and views in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-149** ADM\_TAB\_HISTOGRAMS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(4000)	Column name.
endpoint_number	numeric	Bucket ID of the histogram.
endpoint_value	numeric	Not supported. Its value is <b>NULL</b> .
endpoint_actual_value	character varying(4000)	Actual value of the bucket endpoint.
endpoint_actual_value_raw	raw	Not supported. Its value is <b>NULL</b> .
endpoint_repeat_count	numeric	Not supported. Its value is <b>NULL</b> .
scope	character varying(7)	Not supported. Its value is <b>SHARED</b> .

## 12.3.46 ADM\_TAB\_PARTITIONS

ADM\_TAB\_PARTITIONS displays information about level-1 partitions (including level-2 partitioned tables) in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas. The distributed system does not support level-2 partitions. Therefore, this view does not store level-1 partition information of level-2 partitioned tables.

**Table 12-150** ADM\_TAB\_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.

Name	Type	Description
high_value	text	Limit of a partition. <ul style="list-style-type: none"> <li>For range partitioning and interval partitioning, the upper limit of each partition is displayed.</li> <li>For list partitioning, the value list of each partition is displayed.</li> <li>For hash partitioning, the number of each partition is displayed.</li> </ul>
tablespace_name	name	Tablespace name of the partitioned table.
schema	character varying(64)	Name of a namespace.
composite	character varying(3)	Specifies whether the table is a level-2 partitioned table.
subpartition_count	numeric	Not supported. Its value is <b>NULL</b> .
high_value_length	integer	Length of the binding value expression in a partition.
partition_position	numeric	Position of the partition in the table.
pct_free	numeric	Minimum percentage of available space in a block.
pct_used	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Initial number of transactions. The default value is <b>4</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is <b>128</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Specifies whether changes to the table are logged.
compression	character varying(8)	Actual compression attribute of a partitioned table.
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_row_len	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool allocated to a partitioned block.
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
is_nested	character varying(3)	Specifies whether this partitioned table is a nested partitioned table.
parent_table_partition	character varying(128)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table. The distributed system does not support interval partitioning. Therefore, the value of this parameter is <b>NO</b> .

Name	Type	Description
segment_created	character varying(4)	Specifies whether a partitioned table has segments created or has not been created.
indexing	character varying(4)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(4)	Not supported. Its value is <b>NULL</b> .
inmemory	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
inmemory_service_name	character varying(100)	Not supported. Its value is <b>NULL</b> .
memoptimize_read	character varying(8)	Not supported. Its value is <b>NULL</b> .
memoptimize_write	character varying(8)	Not supported. Its value is <b>NULL</b> .

### 12.3.47 ADM\_TAB\_PRIVS

ADM\_TAB\_PRIVS displays authorization information about all objects in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-151** ADM\_TAB\_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
owner	character varying(128)	Object owner.

Name	Type	Description
table_name	character varying(128)	Name of an object. Object types include tables, packages, indexes, and sequences.
grantor	character varying(128)	Name of the user who grants the permission.
privilege	character varying(40)	Permissions on an object, including USAGE, UPDATE, DELETE, INSERT, CONNECT, SELECT, and EXECUTE.
grantable	character varying(3)	Specifies whether the grant contains the <b>GRANT</b> option. <ul style="list-style-type: none"> <li>• <b>YES</b></li> <li>• <b>NO</b></li> </ul>
type	character varying(24)	Object types, including NODE GROUP, COLUMN_ENCRYPTION_KEY, COLUMN, TABLE, VIEW, SEQUENCE, TYPE, INDEX, DATABASE, DIRECTORY, FOREIGN DATA WRAPPER, FOREIGN SERVER, LANGUAGE, SCHEMA, TEMPLATE, FUNCTION, PROCEDURE, and TABLESPACE.
hierarchy	character varying(3)	Not supported. Its value is <b>NULL</b> .
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.48 ADM\_TAB\_STATISTICS

ADM\_TAB\_STATISTICS displays optimizer statistics for all tables in the database. This view exists in both PG\_CATALOG and SYS schemas. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

**Table 12-152** ADM\_TAB\_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Object owner.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
partition_position	numeric	Not supported. Its value is <b>NULL</b> .
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
subpartition_position	numeric	Not supported. Its value is <b>NULL</b> .
object_type	character varying(12)	Object type. <ul style="list-style-type: none"> <li>• <b>TABLE</b></li> <li>• <b>PARTITION</b></li> <li>• <b>SUBPARTITION</b></li> </ul>
num_rows	numeric	Number of rows in an object.
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_row_len	numeric	Average row length, including the row overhead.
avg_space_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .
num_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_cached_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_cache_hit_ratio	numeric	Not supported. Its value is <b>NULL</b> .
im_imcu_count	numeric	Not supported. Its value is <b>NULL</b> .
im_block_count	numeric	Not supported. Its value is <b>NULL</b> .
im_stat_update_time	timestamp(9) without time zone	Not supported. Its value is <b>NULL</b> .
scan_rate	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Number of samples used for analyzing a table.
last_analyzed	timestamp with time zone	Date when a table was last analyzed. Database restart is not supported. Otherwise, data loss will occur.
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
stattype_locked	character varying(5)	Not supported. Its value is <b>NULL</b> .
stale_stats	character varying(7)	Not supported. Its value is <b>NULL</b> .
notes	character varying(25)	Not supported. Its value is <b>NULL</b> .
scope	character varying(7)	Not supported. The default value is <b>SHARED</b> .

### 12.3.49 ADM\_TAB\_STATS\_HISTORY

ADM\_TAB\_STATS\_HISTORY provides historical statistics about all tables in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-153** ADM\_TAB\_STATS\_HISTORY columns

Name	Type	Description
owner	character varying(128)	Object owner.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
stats_update_time	timestamp(6) with time zone	Time when statistics are updated. Database restart is not supported. Otherwise, data loss will occur.

### 12.3.50 ADM\_TABLES

ADM\_TABLES displays all tables in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-154** ADM\_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
tablespace_name	character varying(64)	Tablespace name of the table.
dropped	character varying	Specifies whether the current table is deleted. <ul style="list-style-type: none"> <li>• <b>YES</b>: It is deleted.</li> <li>• <b>NO</b>: It is not deleted.</li> </ul>
num_rows	numeric	Estimated number of rows in the table.
status	character varying(8)	Specifies whether the current table is valid. <ul style="list-style-type: none"> <li>• <b>VALID</b>: The current table is valid.</li> <li>• <b>UNUSABLE</b>: The current table is unavailable.</li> </ul>
sample_size	numeric	Number of samples used for analyzing the table.
temporary	character(1)	Specifies whether the table is a temporary table. <ul style="list-style-type: none"> <li>• <b>Y</b>: The table is a temporary table.</li> <li>• <b>N</b>: The table is not a temporary table.</li> </ul>
pct_free	numeric	Minimum percentage of free space in a block.
ini_trans	numeric	Initial number of transactions.
max_trans	numeric	Maximum number of transactions.
avg_row_len	integer	Average number of bytes in each row.

Name	Type	Description
partitioned	character varying(3)	Specifies whether a table is a partitioned table. <ul style="list-style-type: none"> <li>● <b>YES:</b> The table is a partitioned table.</li> <li>● <b>NO:</b> The table is not a partitioned table.</li> </ul>
last_analyzed	timestamp with time zone	Last time when the table was analyzed.
row_movement	character varying(8)	Specifies whether to allow partition row movement. <ul style="list-style-type: none"> <li>● <b>ENABLED:</b> The partition row movement is allowed.</li> <li>● <b>DISABLED:</b> The partition row movement is not allowed.</li> </ul>
compression	character varying(8)	Specifies whether to enable a table compression. <ul style="list-style-type: none"> <li>● <b>ENABLED:</b> A table compression is enabled.</li> <li>● <b>DISABLED:</b> A table compression is disabled.</li> </ul>
duration	character varying(15)	Time elapsed when a temporary table is processed. <ul style="list-style-type: none"> <li>● <b>NULL:</b> The table is not a temporary table.</li> <li>● <b>sys\$session:</b> The table is a temporary session table.</li> <li>● <b>sys\$transaction:</b> The table is a temporary transaction table.</li> </ul>

Name	Type	Description
logical_replication	character varying(8)	Specifies whether logical replication is enabled for a table. <ul style="list-style-type: none"> <li>• <b>ENABLED</b>: Logical replication is enabled.</li> <li>• <b>DISABLED</b>: Logical replication is disabled.</li> </ul>
external	character varying(3)	Specifies whether the table is a foreign table. <ul style="list-style-type: none"> <li>• <b>YES</b>: The table is a foreign table.</li> <li>• <b>NO</b>: The table is not a foreign table.</li> </ul>
logging	character varying(3)	Specifies whether changes to a table are logged. <ul style="list-style-type: none"> <li>• <b>YES</b>: Changes to the table are logged.</li> <li>• <b>NO</b>: Changes to the table are not logged.</li> </ul>
default_collation	character varying(100)	Default collation of a table. <ul style="list-style-type: none"> <li>• default</li> </ul>
degree	character varying(10)	Number of instances in a scanned table.
table_lock	character varying(8)	Specifies whether to enable a table lock. <ul style="list-style-type: none"> <li>• <b>ENABLED</b>: The table lock is enabled.</li> <li>• <b>DISABLED</b>: The table lock is disabled.</li> </ul>
buffer_pool	character varying(7)	Default buffer pool of a table.
flash_cache	character varying(7)	Smart flash cache hint in database for a table block.
cell_flash_cache	character varying(7)	Cell flash cache hint for a table block.

Name	Type	Description
skip_corrupt	character varying(8)	Specifies whether to skip corrupted blocks during table scanning. <ul style="list-style-type: none"> <li>● <b>ENABLED:</b> The corrupted block is skipped.</li> <li>● <b>DISABLED:</b> The corrupted block is not skipped.</li> </ul>
has_identity	character varying(3)	Specifies whether a table has an identifier column. <ul style="list-style-type: none"> <li>● <b>YES:</b> There is an identifier column.</li> <li>● <b>NO:</b> There is no identifier column.</li> </ul>
segment_created	character varying(3)	Specifies whether a table segment has been created. <ul style="list-style-type: none"> <li>● <b>YES:</b> The table segment has been created.</li> <li>● <b>NO:</b> The table segment is not created.</li> </ul>
monitoring	character varying(3)	Specifies whether to monitor the modification of a table. <ul style="list-style-type: none"> <li>● <b>YES:</b> The modification of the table is monitored.</li> <li>● <b>NO:</b> The modification of the table is not monitored.</li> </ul>
cluster_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
iot_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
pct_used	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
backed_up	character varying(1)	Not supported. Its value is <b>NULL</b> .
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_space_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .
num_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .
instances	character varying(10)	Not supported. Its value is <b>NULL</b> .
cache	character varying(5)	Not supported. Its value is <b>NULL</b> .
iot_type	character varying(12)	Not supported. Its value is <b>NULL</b> .
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
cluster_owner	character varying(30)	Not supported. Its value is <b>NULL</b> .
dependencies	character varying(8)	Not supported. Its value is <b>NULL</b> .
compression_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(3)	Not supported. Its value is <b>NULL</b> .
result_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
clustering	character varying(3)	Not supported. Its value is <b>NULL</b> .
activity_tracking	character varying(23)	Not supported. Its value is <b>NULL</b> .
dml_timestamp	character varying(25)	Not supported. Its value is <b>NULL</b> .
container_data	character varying(3)	Not supported. Its value is <b>NULL</b> .
inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
duplicated	character varying(1)	Not supported. Its value is <b>NULL</b> .
sharded	character varying(1)	Not supported. Its value is <b>NULL</b> .
hybrid	character varying(3)	Not supported. Its value is <b>NULL</b> .
cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
containers_default	character varying(3)	Not supported. Its value is <b>NULL</b> .
container_map	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
extended_data_link	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .
container_map_object	character varying(3)	Not supported. Its value is <b>NULL</b> .
memoptimize_read	character varying(8)	Not supported. Its value is <b>NULL</b> .
memoptimize_write	character varying(8)	Not supported. Its value is <b>NULL</b> .
has_sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
admit_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_link_dml_enabled	character varying(3)	Not supported. Its value is <b>NULL</b> .
object_id_type	character varying(16)	Not supported. Its value is <b>NULL</b> .
table_type_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
table_type	character varying(128)	Not supported. Its value is <b>NULL</b> .
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .

### 12.3.51 ADM\_TABLESPACES

ADM\_TABLESPACES displays information about available tablespaces. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas. The logical structure features of GaussDB are different from those of the ORA database.

**Table 12-155** ADM\_TABLESPACES columns

Name	Type	Description
tablespace_name	character varying(64)	Tablespace name
block_size	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
min_extlen	numeric	Not supported. Its value is <b>NULL</b> .
contents	character varying(9)	Not supported. Its value is <b>NULL</b> .
status	character varying(9)	Not supported The default value is <b>ONLINE</b> .
logging	character varying(9)	Not supported. Its value is <b>NULL</b> .
force_logging	character varying(3)	Not supported. Its value is <b>NULL</b> .
extent_management	character varying(10)	Not supported. Its value is <b>NULL</b> .
allocation_type	character varying(9)	Not supported. Its value is <b>NULL</b> .
plugged_in	character varying(3)	Not supported. Its value is <b>NULL</b> .
segment_space_management	character varying(6)	Not supported. Its value is <b>NULL</b> .
def_tab_compression	character varying(8)	Not supported. Its value is <b>NULL</b> .
retention	character varying(11)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
bigfile	character varying(3)	Not supported. Its value is <b>NULL</b> .
predicate_evaluation	character varying(7)	Not supported. Its value is <b>NULL</b> .
encrypted	character varying(3)	Not supported. Its value is <b>NULL</b> .
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
def_inmemory	character varying(8)	Not supported. Its value is <b>NULL</b> .
def_inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
def_inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
def_inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
def_inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
shared	character varying(12)	Not supported. Its value is <b>NULL</b> .
def_index_compression	character varying(8)	Not supported. Its value is <b>NULL</b> .
index_compress_for	character varying(13)	Not supported. Its value is <b>NULL</b> .
def_cellmemory	character varying(14)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .
lost_write_protect	character varying(7)	Not supported. Its value is <b>NULL</b> .
chunk_tablespace	character varying(1)	Not supported. Its value is <b>NULL</b> .

## 12.3.52 ADM\_TRIGGERS

ADM\_TRIGGERS displays information about triggers in the database. By default, only the system administrator can access this view. Common users can access the

view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-156** ADM\_TRIGGERS columns

Name	Type	Description
owner	character varying(128)	Trigger owner.
trigger_name	character varying(64)	Trigger name.
trigger_type	character varying	Trigger types: <b>before statement, before each row, after statement, after each row, and instead of.</b>
triggering_event	character varying	Events that trigger a trigger: <b>update, insert, delete, and truncate.</b>
table_owner	character varying(64)	Owner of the table that defines a trigger.
base_object_type	character varying(18)	Basic object type of a trigger, which can be <b>table</b> or <b>view</b> .
table_name	character varying(64)	Name of the table or view that defines a trigger.
column_name	character varying(4000)	Not supported. Its value is <b>NULL</b> .
referencing_name	character varying(422)	Not supported. Its value is <b>referencing new as new old as old</b> .
when_clause	character varying(4000)	Content of the WHEN clause. <b>trigger_body</b> can be executed only when the value is <b>true</b> .
status	character varying(64)	<ul style="list-style-type: none"><li>● <b>O</b>: The trigger is enabled in origin or local mode.</li><li>● <b>D</b>: The trigger is disabled.</li><li>● <b>R</b>: The trigger is enabled in replica mode.</li><li>● <b>A</b>: The trigger is always enabled.</li></ul>
description	character varying(4000)	Trigger description, which is used to rebuild a trigger creation statement.
action_type	character varying(11)	Action type of a trigger, which only supports <b>call</b> .
trigger_body	text	Statement executed when a trigger is triggered.
crossedition	character varying(7)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
before_statement	character varying(3)	Not supported. Its value is <b>NULL</b> .
before_row	character varying(3)	Not supported. Its value is <b>NULL</b> .
after_row	character varying(3)	Not supported. Its value is <b>NULL</b> .
after_statement	character varying(3)	Not supported. Its value is <b>NULL</b> .
instead_of_row	character varying(3)	Not supported. Its value is <b>NULL</b> .
fire_once	character varying(3)	Not supported. Its value is <b>NULL</b> .
apply_server_only	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.53 ADM\_TYPES

ADM\_TYPES describes all object types in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-157** ADM\_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of the type.
type_name	character varying(128)	Type name.
type_oid	raw	Type OID.
typecode	character varying(128)	Type code.
attributes	numeric	Number of attributes in a type.
methods	numeric	Not supported. Its value is <b>0</b> .
predefined	character varying(3)	Specifies whether the type is a predefined type.
incomplete	character varying(3)	Specifies whether the type is an incomplete type.
final	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
instantiable	character varying(3)	Not supported. Its value is <b>NULL</b> .
persistable	character varying(3)	Not supported. Its value is <b>NULL</b> .
supertype_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
supertype_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
local_attributes	numeric	Not supported. Its value is <b>NULL</b> .
local_methods	numeric	Not supported. Its value is <b>NULL</b> .
typeid	raw	Not supported. Its value is <b>NULL</b> .

## 12.3.54 ADM\_TYPE\_ATTRS

ADM\_TYPE\_ATTRS displays the attributes of the current database object type. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-158** ADM\_TYPE\_ATTRS columns

Name	Type	Description
owner	oid	Owner of the type.
type_name	name	Data type name.
attr_name	name	Column name.
attr_type_mod	integer	Type-specific data supplied at the table creation time (for example, the maximum length of a <b>varchar</b> column). This column is used as the third parameter when passing to type-specific input functions and length coercion functions. The value will generally be <b>-1</b> for types that do not need ATTTYPMOD.
attr_type_owner	oid	Owner of an attribute of this type.
attr_type_name	name	Data type attribute name. Currently, this column records the type name after conversion.

Name	Type	Description
length	smallint	Number of bytes in the internal representation of the type for a fixed-size type. It is a negative number for a variable-length type. <ul style="list-style-type: none"> <li>• The value <b>-1</b> indicates a "varlena" type (one that has a length word).</li> <li>• The value <b>-2</b> indicates a NULL-terminated C string.</li> </ul>
precision	integer	Precision of the numeric type.
scale	integer	Range of the numeric type.
character_set_name	character(1)	Character set name of an attribute ( <b>c</b> or <b>n</b> ). <ul style="list-style-type: none"> <li>• <b>c</b>: CHAR_CS.</li> <li>• <b>n</b>: NCHAR_CS.</li> </ul>
attr_no	smallint	Attribute number
inherited	character(1)	Specifies whether the attribute is inherited from the super type ( <b>Y</b> or <b>N</b> )
attr_length	integer	Type-specific data supplied at the table creation time (for example, the maximum length of a <b>varchar</b> column). The raw type is not recorded due to kernel implementation.

### 12.3.55 ADM\_USERS

ADM\_USERS displays information about all database users. This view is accessible only to system administrators. This view exists in both the PG\_CATALOG and SYS schemas.

**Table 12-159** ADM\_USERS columns

Name	Type	Description
username	character varying(128)	Username.
user_id	oid	User ID.

Name	Type	Description
account_status	character varying(32)	Account status. <ul style="list-style-type: none"> <li>● <b>null</b>: The account is the initial system administrator with the highest permission.</li> <li>● <b>0</b>: normal.</li> <li>● <b>1</b>: The account is locked for a specific period of time because the number of failed login attempts exceeds the threshold.</li> <li>● <b>2</b>: The account is locked by the administrator.</li> </ul>
lock_date	timestamp with time zone	By default, the creation date of the account is displayed. If the account is locked by the administrator or the account is locked because the number of login failures exceeds the threshold, the date when the account is locked is displayed. The value is <b>null</b> for the initial system administrator.
expiry_date	timestamp with time zone	Account expiration date.
default_tablespace	character varying(4000)	Default tablespace for storing data.
temporary_tablespace	character varying(4000)	Name of the default tablespace or tablespace group of a temporary table.
local_temp_tablespace	character varying(30)	Not supported. The default value is <b>NULL</b> .
created	timestamp with time zone	Date when a user is created.
profile	character varying(128)	Not supported. The default value is <b>NULL</b> .
initial_rsrc_consumer_group	character varying(128)	Not supported. The default value is <b>NULL</b> .
external_name	character varying(4000)	Not supported. The default value is <b>NULL</b> .
password_versions	character varying(12)	Encryption mode of the account password. The value can be <b>MD5</b> , <b>SHA256</b> , or <b>SM3</b> .

Name	Type	Description
editions_enabled	character varying(1)	Not supported. The default value is <b>NULL</b> .
authentication_type	text	Indicates the authentication mechanism of the user.
proxy_only_connect	character varying(1)	Not supported. The default value is <b>NULL</b> .
common	character varying(3)	Not supported. The default value is <b>NULL</b> .
last_login	timestamp with time zone	Last login.
oracle_maintained	character varying(1)	Not supported. The default value is <b>NULL</b> .
inherited	character varying(3)	Not supported. The default value is <b>NULL</b> .
default_collation	character varying(100)	Default collation of the user schema.
implicit	character varying(3)	Not supported. The default value is <b>NULL</b> .
all_shard	character varying(3)	Not supported. The default value is <b>NULL</b> .
password_change_date	timestamp with time zone	Date when the user password was set last time.

### 12.3.56 ADM\_VIEWS

ADM\_VIEWS displays views in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-160** ADM\_VIEWS columns

Name	Type	Description
owner	character varying(64)	Owner of a view.
view_name	character varying(64)	View name.
text	text	Text in the view.
text_length	integer	Text length of the view.

Name	Type	Description
text_vc	character varying(4000)	View creation statement. This column may truncate the view text. The BEQUEATH clause will not appear as part of the <b>TEXT_VC</b> column in this view.
type_text_length	numeric	Not supported. Its value is <b>NULL</b> .
type_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .
oid_text_length	numeric	Not supported. Its value is <b>NULL</b> .
oid_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .
view_type_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
view_type	character varying(128)	Not supported. Its value is <b>NULL</b> .
superview_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
editioning_view	character varying(1)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(1)	Not supported. Its value is <b>NULL</b> .
container_data	character varying(1)	Not supported. Its value is <b>NULL</b> .
bequeath	character varying(12)	Not supported. Its value is <b>NULL</b> .
origin_con_id	character varying(256)	Not supported. Its value is <b>NULL</b> .
default_collation	character varying(100)	Not supported. Its value is <b>NULL</b> .
containers_default	character varying(3)	Not supported. Its value is <b>NULL</b> .
container_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
extended_data_link_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
has_sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
admit_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
pdb_local_only	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.57 COMM\_CLIENT\_INFO

COMM\_CLIENT\_INFO displays information about active client connections of a single node (You can query this view on a DN to view the information about the connection between the CN and DN). By default, only the users with system administrator permission can access this view.

**Table 12-161** COMM\_CLIENT\_INFO columns

Name	Type	Description
node_name	text	Name of the current DN, for example, dn_6001_6002_6003.
app	text	Query a view on a DN. The app displays the client connected to the current DN, such as the coordinator (CN), GTM, or DN.
tid	bigint	Thread ID of the current thread.
lwtid	integer	Lightweight thread ID of the current thread.
query_id	bigint	Query ID. It is equivalent to <b>debug_query_id</b> .
socket	integer	<b>socket fd</b> is displayed if the connection is a physical connection.
remote_ip	text	Peer node IP address.
remote_port	text	Peer node port.
logic_id	integer	Displayed if the connection is a logical connection.

### 12.3.58 DB\_ALL\_TABLES

DB\_ALL\_TABLES displays tables or views accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-162** DB\_ALL\_TABLES columns

Name	Type	Description
owner	name	Owner of a table or view.
table_name	name	Name of the table or view.
tablespace_name	name	Tablespace where the table or view is located.

## 12.3.59 DB\_ARGUMENTS

DB\_ARGUMENTS displays parameter information about stored procedures and functions accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas. This view is accessible to all users and displays all information accessible to the current user.

**Table 12-163** DB\_ARGUMENTS columns

Name	Type	Description
owner	character varying(128)	Owner of a function or stored procedure.
object_name	character varying(128)	Name of a function or stored procedure.
object_id	oid	OID of a function or stored procedure.
overload	character varying(40)	<i>n</i> th overloaded function of the name.
subprogram_id	numeric	Not supported. Its value is <b>NULL</b> .
argument_name	character varying(128)	Parameter name.
position	numeric	Position of the parameter in the parameter list. The value is <b>0</b> for the return value of the function by default.
sequence	numeric	Sequence of a parameter, which starts from 1, with the return type before all parameters.
data_level	numeric	Nesting depth of parameters of the composite type. The value is fixed at <b>0</b> because only one line is displayed for each parameter.
data_type	character varying(64)	Data type of a parameter.

Name	Type	Description
defaulted	character varying(1)	Specifies whether a parameter has a default value. <ul style="list-style-type: none"> <li>• <b>Y</b>: yes.</li> <li>• <b>N</b>: no.</li> </ul>
default_value	text	Not supported. Its value is <b>NULL</b> .
default_length	numeric	Not supported. Its value is <b>NULL</b> .
in_out	character varying(9)	Input and output attributes of a parameter: <ul style="list-style-type: none"> <li>• <b>IN</b>: input parameter.</li> <li>• <b>OUT</b>: output parameter.</li> <li>• <b>IN_OUT</b>: input and output parameters.</li> <li>• <b>VARIADIC</b>: VARIADIC parameter.</li> </ul>
data_length	numeric	Not supported. Its value is <b>NULL</b> .
data_precision	numeric	Not supported. Its value is <b>NULL</b> .
data_scale	numeric	Not supported. Its value is <b>NULL</b> .
radix	numeric	Radix of a number. Set it to <b>10</b> for the smallint, integer, bigint, numeric, or float type, and to <b>NULL</b> for other data types.
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
type_owner	character varying(128)	Owner of a data type.
type_name	character varying(128)	Parameter type name.
type_subname	character varying(128)	Not supported. Its value is <b>NULL</b> .
type_link	character varying(128)	Not supported. Its value is <b>NULL</b> .
type_object_type	character varying(7)	Types of types of the <b>type_owner</b> , <b>type_name</b> , and <b>type_subname</b> columns: <ul style="list-style-type: none"> <li>• <b>TABLE</b>: The parameter is of the table type.</li> <li>• <b>VIEW</b>: The parameter is of the view type.</li> <li>• Set this column to <b>NULL</b> for other types.</li> </ul>

Name	Type	Description
pls_type	character varying(128)	Name of the PL/SQL type for parameters of the number type. Otherwise, this column is empty.
char_length	numeric	Not supported. Its value is <b>NULL</b> .
char_used	character varying(1)	Not supported. Set it to <b>B</b> if the data type is varchar, nvarchar2, bpchar, or char, and to <b>NULL</b> for other data types.
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .

### 12.3.60 DB\_COL\_COMMENTS

DB\_COL\_COMMENTS displays comment information about table columns accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-164** DB\_COL\_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	character varying(64)	Column name.
comments	text	Comments.
schema	character varying(64)	Name of the namespace to which the column belongs.

### 12.3.61 DB\_COL\_PRIVS

DB\_COL\_PRIVS displays the following granting information:

- Column permission granting information when the current user is the object owner, grantor, or grantee.
- Column permission granting information when the enabled role or PUBLIC role is the grantee.

By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-165** DB\_COL\_PRIVS columns

Name	Type	Description
grantor	character varying(128)	Name of the user who grants the permission.
owner	character varying(128)	Object owner.
grantee	character varying(128)	Name of the user or role to which the permission is granted.
table_schema	character varying(128)	Schema of an object.
table_name	character varying(128)	Object name.
column_name	character varying(128)	Column name.
privilege	character varying(40)	Permission on a column.
grantable	character varying(3)	Specifies whether to grant privileges. <ul style="list-style-type: none"> <li>• YES</li> <li>• NO</li> </ul>
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

## 12.3.62 DB\_COLL\_TYPES

DB\_COLL\_TYPES displays information about collection types that can be accessed by the current user. By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-166** DB\_COLL\_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of a cluster.
type_name	character varying(128)	Name of a collection.
coll_type	character varying(128)	Description of a set.
upper_bound	numeric	Not supported. Its value is <b>NULL</b> .
elem_type_mod	character varying(7)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
elem_type_name	character varying(128)	Name of the data type or user-defined type on which the collection is based.
length	numeric	Not supported. Its value is <b>NULL</b> .
precision	numeric	Not supported. Its value is <b>NULL</b> .
scale	numeric	Not supported. Its value is <b>NULL</b> .
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
elem_storage	character varying(7)	Not supported. Its value is <b>NULL</b> .
nulls_stored	character varying(3)	Not supported. Its value is <b>NULL</b> .
char_used	character varying(1)	Not supported. Its value is <b>NULL</b> .

### 12.3.63 DB\_CONSTRAINTS

DB\_CONSTRAINTS displays information about constraints accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-167** DB\_CONSTRAINTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
constraint_name	character varying(64)	Constraint name.
constraint_type	text	Constraint type. <ul style="list-style-type: none"> <li>● <b>c</b>: check constraint.</li> <li>● <b>f</b>: foreign key constraint.</li> <li>● <b>p</b>: primary key constraint.</li> <li>● <b>u</b>: unique constraint.</li> </ul>
table_name	character varying(64)	Name of a constraint-related table.
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint).
index_name	character varying(64)	Name of the constraint-related index (only for the unique constraint and primary key constraint).

## 12.3.64 DB\_CONS\_COLUMNS

DB\_CONS\_COLUMNS displays information about constraint columns accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-168** DB\_CONS\_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
constraint_name	character varying(64)	Constraint name.
table_name	character varying(64)	Name of a constraint-related table.
column_name	character varying(64)	Name of a constraint-related column.
position	smallint	Position of a column in a table.

## 12.3.65 DB\_DEPENDENCIES

DB\_DEPENDENCIES displays the dependency relationship between types, tables, views, stored procedures, functions, and triggers accessible to the current user. It is accessible to all users. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-169** DB\_DEPENDENCIES columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	character varying(18)	Object type.
referenced_owner	name	Owner of the referenced object.
referenced_name	name	Name of the referenced object.
referenced_type	character varying(18)	Type of the referenced object.
referenced_link_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
dependency_type	character varying(4)	Not supported. Its value is <b>NULL</b> .

## 12.3.66 DB\_IND\_COLUMNS

**DB\_IND\_COLUMNS** displays all index columns accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-170** DB\_IND\_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	name	Column name
column_position	smallint	Position of the column in the index
column_length	numeric	Column length
char_length	numeric	Length of a column (character type).
descend	character varying	Indicates the sorting mode of a column. The value can be <b>DESC</b> or <b>ASC</b> .
collated_column_id	numeric	Provides the internal serial number of the language sorting columns.

## 12.3.67 DB\_IND\_EXPRESSIONS

**DB\_IND\_EXPRESSIONS** displays information about expression indexes accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-171** DB\_IND\_EXPRESSIONS columns

Name	Type	Description
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.

Name	Type	Description
column_expression	text	Function-based index expression of a specified column.
column_position	smallint	Position of the column in the index.

### 12.3.68 DB\_IND\_PARTITIONS

DB\_IND\_PARTITIONS displays partition information about local indexes of level-1 partitioned table accessible to the current user (excluding global indexes of partitioned tables). It is accessible to all users. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-172** DB\_IND\_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index to which an index partition belongs.
index_name	character varying(64)	Name of the partitioned table index to which the index partition belongs.
partition_name	character varying(64)	Name of the index partition.
def_tablespace_name	name	Tablespace name of the index partition.
high_value	text	Upper limit of the partition corresponding to the index partition. <ul style="list-style-type: none"> <li>For range partitioning and interval partitioning, the upper limit of each partition is displayed.</li> <li>For list partitioning, the value list of each partition is displayed.</li> <li>For hash partitioning, the number of each partition is displayed.</li> </ul>

Name	Type	Description
index_partition_usable	Boolean	Specifies whether the index partition is available. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
schema	character varying(64)	Schema of the partitioned table index to which the index partition belongs.
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition
composite	character varying(3)	Specifies whether the index is a local index on the level-2 partitioned table. This table does not store level-2 partition information. Therefore, the value is <b>NO</b> .
subpartition_count	numeric	Number of level-2 partitions in a partition. This table does not store level-2 partition information. Therefore, the value is <b>0</b> .
partition_position	numeric	Position of an index partition in the index.
status	character varying(8)	Specifies whether the index partition is available.
tablespace_name	name	Name of the tablespace that contains the partition.
pct_free	numeric	Percentage of minimum available space in a block.
ini_trans	numeric	Initial number of transactions. The default value is <b>4</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is <b>128</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Specifies whether changes to an index are logged.
compression	character varying(13)	Specifies whether an index compression is enabled for a partitioned index.
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
clustering_factor	numeric	Sequence of a row in the table based on the value of the index. You need to run the <b>analyze</b> command to collect statistics.
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Actual buffer pool of a partition.
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
pct_direct_access	numeric	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table. The distributed system does not support interval partitioning. Therefore, the value of this parameter is <b>NO</b> .
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
orphaned_entries	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.69 DB\_IND\_SUBPARTITIONS

DB\_IND\_SUBPARTITIONS displays partition information about local indexes of level-2 partitioned table accessible to the current user (excluding global indexes of partitioned tables). It is accessible to all users. This view exists in both PG\_CATALOG and SYS schemas. The distributed system does not support level-2 partitions. Currently, all columns in this table are set to **NULL**.

**Table 1** DB\_IND\_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
index_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
high_value	text	Not supported. Its value is <b>NULL</b> .
high_value_length	numeric	Not supported. Its value is <b>NULL</b> .
partition_position	numeric	Not supported. Its value is <b>NULL</b> .
subpartition_position	numeric	Not supported. Its value is <b>NULL</b> .
status	character varying(8)	Not supported. Its value is <b>NULL</b> .
tablespace_name	character varying(30)	Not supported. Its value is <b>NULL</b> .
pct_free	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Not supported. Its value is <b>NULL</b> .
max_trans	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Not supported. Its value is <b>NULL</b> .
compression	character varying(13)	Not supported. Its value is <b>NULL</b> .
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
clustering_factor	numeric	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .
buffer_pool	character varying(7)	Not supported. Its value is <b>NULL</b> .
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
segment_created	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .

## 12.3.70 DB\_INDEXES

DB\_INDEXES displays information about indexes accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-173** DB\_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_name	character varying(64)	Name of the table corresponding to the index.
uniqueness	text	Specifies whether the index is a unique index. <ul style="list-style-type: none"> <li>• <b>UNIQUE</b>: unique index.</li> <li>• <b>NONUNIQUE</b>: non-unique index.</li> </ul>
partitioned	character(3)	Specifies whether the index has the property of partitioned tables. <ul style="list-style-type: none"> <li>• <b>Yes</b>: The index has the property of a partitioned table.</li> <li>• <b>No</b>: The index does not have the property of a partitioned table.</li> </ul>

Name	Type	Description
generated	character varying(1)	Specifies whether the index name is generated by the system. <ul style="list-style-type: none"> <li>• <b>y</b>: The index name is generated by the system.</li> <li>• <b>n</b>: The index name is not generated by the system.</li> </ul>
index_type	character varying(27)	Index type. <ul style="list-style-type: none"> <li>• <b>NORMAL</b>: Index attributes are simple references, and the expression tree is empty.</li> <li>• <b>FUNCTION-BASED NORMAL</b>: Expression trees are used for index attributes that are not simple column references.</li> </ul>
table_owner	character varying(128)	Owner of an index object.
table_type	character(11)	Type of the index object. <ul style="list-style-type: none"> <li>• <b>TABLE</b>: The index object is of the table type.</li> </ul>
tablespace_name	character varying(30)	Name of a tablespace that contains the index.
status	character varying(8)	Status of a non-partitioned index. <ul style="list-style-type: none"> <li>• <b>VALID</b>: Non-partitioned indexes can be used for query.</li> <li>• <b>UNUSABLE</b>: The non-partitioned index is unavailable.</li> <li>• <b>N/A</b>: The index has the property of a partitioned table.</li> </ul>
compression	character varying(13)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
prefix_length	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Not supported. Its value is <b>NULL</b> .
max_trans	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
pct_threshold	numeric	Not supported. Its value is <b>NULL</b> .
include_column	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
pct_free	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(3)	Not supported. Its value is <b>NULL</b> .
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
clustering_factor	numeric	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .
degree	character varying(40)	Not supported. Its value is <b>NULL</b> .
instances	character varying(40)	Not supported. Its value is <b>NULL</b> .
temporary	character varying(1)	Not supported. Its value is <b>NULL</b> .
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
buffer_pool	character varying(7)	Not supported. Its value is <b>NULL</b> .
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
duration	character varying(15)	Not supported. Its value is <b>NULL</b> .
pct_direct_access	numeric	Not supported. Its value is <b>NULL</b> .
ityp_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
ityp_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_status	character varying(12)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
funcidx_status	character varying(8)	Not supported. Its value is <b>NULL</b> .
join_index	character varying(3)	Not supported. Its value is <b>NULL</b> .
iot_redundant_pkey_elim	character varying(3)	Not supported. Its value is <b>NULL</b> .
dropped	character varying(3)	Not supported. Its value is <b>NULL</b> .
visibility	character varying(9)	Not supported. Its value is <b>NULL</b> .
domidx_management	character varying(14)	Not supported. Its value is <b>NULL</b> .
segment_created	character varying(3)	Not supported. Its value is <b>NULL</b> .
orphaned_entries	character varying(3)	Not supported. Its value is <b>NULL</b> .
indexing	character varying(7)	Not supported. Its value is <b>NULL</b> .
auto	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.71 DB\_OBJECTS

DB\_OBJECTS displays all database objects accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-174** DB\_OBJECTS columns

Name	Type	Description
owner	name	Object owner.
object_name	name	Object name.
object_id	oid	Object OID.
object_type	name	Object type.
namespace	oid	ID of the namespace where the object resides.

Name	Type	Description
temporary	character(1)	Specifies whether the object is a temporary object.
status	character varying(7)	Object status.
subobject_name	name	Subobject name of the object.
generated	character(1)	Specifies whether the object name is generated by the system.
created	timestamp with time zone	Time when the object was created.
last_ddl_time	timestamp with time zone	Last time when the object was modified.
default_collation	character varying(100)	Default collation of objects.
data_object_id	numeric	Not supported. Its value is <b>NULL</b> .
timestamp	character varying(19)	Not supported. Its value is <b>NULL</b> .
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
edition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
sharing	character varying(18)	Not supported. Its value is <b>NULL</b> .
editionable	character varying(1)	Not supported. Its value is <b>NULL</b> .
oracle_maintained	character varying(1)	Not supported. Its value is <b>NULL</b> .
application	character varying(1)	Not supported. Its value is <b>NULL</b> .
duplicated	character varying(1)	Not supported. Its value is <b>NULL</b> .
sharded	character varying(1)	Not supported. Its value is <b>NULL</b> .
created_appid	numeric	Not supported. Its value is <b>NULL</b> .
modified_appid	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
created_vsnid	numeric	Not supported. Its value is <b>NULL</b> .
modified_vsnid	numeric	Not supported. Its value is <b>NULL</b> .

**NOTICE**

For details about the value ranges of **created** and **last\_ddl\_time**, see [PG\\_OBJECT](#).

## 12.3.72 DB\_PART\_COL\_STATISTICS

DB\_PART\_COL\_STATISTICS displays column statistics and histogram information about table partitions accessible to the current user. All users can access this view. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-175** DB\_PART\_COL\_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Owner of the partitioned table.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Table partition name.
column_name	character varying(4000)	Column name.
num_distinct	numeric	Not supported. Its value is <b>NULL</b> .
low_value	raw	Not supported. Its value is <b>NULL</b> .
high_value	raw	Not supported. Its value is <b>NULL</b> .
density	numeric	Not supported. Its value is <b>NULL</b> .
num_nulls	numeric	Not supported. Its value is <b>NULL</b> .
num_buckets	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
notes	character varying(63)	Not supported. Its value is <b>NULL</b> .
avg_col_len	numeric	Not supported. Its value is <b>NULL</b> .
histogram	character varying(15)	Not supported. Its value is <b>NULL</b> .
schema	character varying(64)	Name of the namespace to which the column belongs.

### 12.3.73 DB\_PART\_KEY\_COLUMNS

DB\_PART\_KEY\_COLUMNS displays information about the partition key columns of partitioned tables or partitioned indexes accessible to the current user. This view is accessible to all users and displays all information accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-176** DB\_PART\_KEY\_COLUMNS columns

Name	Type	Description
owner	character varying(128)	Owner of a partitioned table or index.
name	character varying(128)	Name of a partitioned table or index.
object_type	character varying(128)	Valid values are <b>table</b> and <b>index</b> . If the partition is a partitioned table, the value is <b>table</b> . If the partition is a partitioned index, the value is <b>index</b> .
column_name	character varying(4000)	Partition key column name of a partitioned table or index.
column_position	numeric	Position of a column in a partition.
collated_column_id	numeric	Not supported. Its value is <b>NULL</b> .

## 12.3.74 DB\_PART\_TABLES

DB\_PART\_TABLES displays information about partitioned tables accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-177** DB\_PART\_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table.
table_name	character varying(64)	Name of the partitioned table.
partitioning_type	text	Partitioning policy of the partitioned table. <b>NOTE</b> For details about the supported partitioning policies of the current partitioned table, see <a href="#">CREATE TABLE PARTITION</a> .
partition_count	bigint	Number of partitions of the partitioned table.
partitioning_key_count	integer	Number of partition keys of the partitioned table.
def_tablespace_name	name	Tablespace name of the partitioned table.
schema	character varying(64)	Schema of the partitioned table.
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, <b>NONE</b> is displayed. The distributed system does not support level-2 partitions. Set this parameter to <b>NONE</b> .
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is <b>1</b> for a level-2 partitioned table and <b>0</b> for a level-1 partitioned table. The distributed system does not support level-2 partitions. Set this parameter to <b>0</b> .
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table. The distributed system does not support level-2 partitions. Set this parameter to <b>0</b> .
status	character varying(8)	Not supported. Its value is <b>valid</b> .

Name	Type	Description
def_pct_free	numeric	Default value of <b>PCTFREE</b> used when a partition is added.
def_pct_used	numeric	Not supported. Its value is <b>NULL</b> .
def_ini_trans	numeric	Default value of <b>INITRANS</b> used when a partition is added.
def_max_trans	numeric	Default value of <b>MAXTRANS</b> used when a partition is added.
def_initial_extent	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_next_extent	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_min_extents	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_max_extents	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_max_size	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_pct_increase	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_freelists	numeric	Not supported. Its value is <b>NULL</b> .
def_freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
def_logging	character varying(7)	Not supported. Its value is <b>NULL</b> .
def_compression	character varying(8)	Default compression mode used when a partition is added. Value range: <ul style="list-style-type: none"><li>• NONE - Not specified</li><li>• ENABLED</li><li>• DISABLED</li></ul>
def_compress_for	character varying(30)	Default compression mode used when a partition is added. <b>NOTE</b> For available compression modes and compression levels, see <a href="#">WITH ( { storage_paramet...</a>
def_buffer_pool	character varying(7)	Not supported. Its value is <b>DEFAULT</b> .
def_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
def_cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
ref_ptn_constraint_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
interval	character varying(1000)	Interval.
autolist	character varying(3)	Not supported. Its value is <b>NO</b> .
interval_subpartition	character varying(1000)	Not supported. Its value is <b>NULL</b> .
autolist_subpartition	character varying(3)	Not supported. Its value is <b>NO</b> .
is_nested	character varying(3)	Not supported. Its value is <b>NO</b> .
def_segment_creation	character varying(4)	Currently, the segment-page mode is not supported. When the segment-page mode is enabled, the value is <b>YES</b> .
def_indexing	character varying(3)	Not supported. Its value is <b>ON</b> .
def_inmemory	character varying(8)	Not supported. Its value is <b>NONE</b> .
def_inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
def_inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
def_inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
def_inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
def_read_only	character varying(3)	Not supported. Its value is <b>NO</b> .
def_cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .

## 12.3.75 DB\_PROCEDURES

DB\_PROCEDURES displays information about all stored procedures or functions accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-178** DB\_PROCEDURES columns

Name	Type	Description
owner	name	Object owner.
object_name	name	Object name.

## 12.3.76 DB\_SCHEDULER\_JOB\_ARGS

DB\_SCHEDULER\_JOB\_ARG displays the parameters related to the tasks accessible to the current user. This view is accessible to all users and displays all information accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-179** DB\_SCHEDULER\_JOB\_ARGS columns

Name	Type	Description
owner	character varying(128)	Owner of the job to which the parameter belongs.
job_name	character varying(128)	Name of the job to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter, which can be a user-defined data type.
value	character varying(4000)	Parameter value.
anydata_value	character varying(4000)	Not supported. Its value is <b>NULL</b> .
out_argument	character varying(5)	Reserved column. Its value is <b>NULL</b> .

## 12.3.77 DB\_SCHEDULER\_PROGRAM\_ARGS

DB\_SCHEDULER\_PROGRAM\_ARG displays the parameters related to the programs accessible to the current user. This view is accessible to all users and displays all information accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-180** DB\_SCHEDULER\_PROGRAM\_ARGS columns

Name	Type	Description
owner	character varying(128)	Owner of the program to which the parameter belongs.
program_name	character varying(128)	Name of the program to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter, which can be a user-defined data type.
metadata_attribute	character varying(19)	Not supported. Its value is <b>NULL</b> .
default_value	character varying(4000)	Default parameter value.
anydata_default_value	character varying(4000)	Not supported. Its value is <b>NULL</b> .
out_argument	character varying(5)	Reserved column. Its value is <b>NULL</b> .

## 12.3.78 DB\_SEQUENCES

DB\_SEQUENCES displays all sequences accessible to the current user. This view exists in both PG\_CATALOG and SYS schemas.

**Table 12-181** DB\_SEQUENCES columns

Name	Type	Description
sequence_owner	name	Owner of a sequence.
sequence_name	name	Name of the sequence.
min_value	int16	Minimum value of the sequence.
max_value	int16	Maximum value of the sequence.

Name	Type	Description
last_number	int16	Value of the previous sequence.
cache_size	int16	Size of the sequence disk cache.
increment_by	int16	Value by which the sequence is incremented.
cycle_flag	character(1)	Specifies whether the sequence is a cyclic sequence. The value can be <b>Y</b> or <b>N</b> . <ul style="list-style-type: none"> <li>• <b>Y</b>: It is a cyclic sequence.</li> <li>• <b>N</b>: It is not a cyclic sequence.</li> </ul>
order_flag	character varying(1)	Specifies whether a sequence occurs in a request sequence. Not supported. Its value is <b>NULL</b> .
scale_flag	character varying(1)	Specifies whether a sequence is a scalable sequence. Not supported. Its value is <b>NULL</b> .
extend_flag	character varying(1)	Specifies whether the value generated by a scalable sequence exceeds the maximum or minimum value of the sequence. Not supported. Its value is <b>NULL</b> .
sharded_flag	character varying(1)	Specifies whether a sequence is a shard sequence. Not supported. Its value is <b>NULL</b> .
session_flag	character varying(1)	Specifies whether a sequence is a private session. Not supported. Its value is <b>NULL</b> .
keep_value	character varying(1)	Specifies whether to retain the sequence value during replay after a failure. Not supported. Its value is <b>NULL</b> .

## 12.3.79 DB\_SOURCE

DB\_SOURCE displays the definition information about stored procedures, functions, and triggers accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-182** DB\_SOURCE columns

Name	Type	Description
owner	name	Object owner.

Name	Type	Description
name	name	Object name.
type	name	Object type. The value can be <b>function</b> , <b>procedure</b> , or <b>trigger</b> .
line	numeric	Number of the line in the definition information.
text	text	Text source of the storage object.
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .

### 12.3.80 DB\_SUBPART\_KEY\_COLUMNS

DB\_SUBPART\_KEY\_COLUMNS displays information about the partition key columns of level-2 partitioned tables or partitioned indexes accessible to the current user. This view is accessible to all users and displays all information accessible to the current user. This view exists in both PG\_CATALOG and SYS schema. The distributed system does not support level-2 partitioned tables. All columns in this view are set to **NULL**.

**Table 12-183** DB\_SUBPART\_KEY\_COLUMNS columns

Name	Type	Description
owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
name	character varying(128)	Not supported. Its value is <b>NULL</b> .
object_type	character varying(128)	Not supported. Its value is <b>NULL</b> .
column_name	character varying(4000)	Not supported. Its value is <b>NULL</b> .
column_position	numeric	Not supported. Its value is <b>NULL</b> .
collated_column_id	numeric	Not supported. Its value is <b>NULL</b> .

### 12.3.81 DB\_SYNONYMS

DB\_SYNONYMS displays all synonyms accessible to the current user.

**Table 12-184** DB\_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym.
schema_name	text	Name of the schema to which the synonym belongs.
synonym_name	text	Synonym name.
table_owner	text	Owner of the associated object. Although the column is called <b>table_owner</b> , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_name	text	Name of the associated object. Although the column is called <b>table_name</b> , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_schema_name	text	Schema name of the associated object. Although the column is called <b>table_schema_name</b> , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
db_link	character varying(128)	Reserved column. Its value is <b>NULL</b> .
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .

### 12.3.82 DB\_TAB\_COL\_STATISTICS

DB\_TAB\_COL\_STATISTICS displays column statistics and histogram information extracted from DB\_TAB\_COLUMNS. All users can access this view. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-185** DB\_TAB\_COL\_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.

Name	Type	Description
num_distinct	numeric	Number of different values in a column.
low_value	raw	Low value in a column.
high_value	raw	High value in a column.
density	numeric	<ul style="list-style-type: none"> <li>If there is a histogram on <b>COLUMN_NAME</b>, this column displays the selectivity of values in the histogram that span less than two endpoints. It does not represent the selectivity of values that span two or more endpoints.</li> <li>If no histogram is available on <b>COLUMN_NAME</b>, the value of this column is <b>1/NUM_DISTINCT</b>.</li> </ul>
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
sample_size	numeric	Sample size used to analyze a column.
last_analyzed	timestamp(0) without time zone	Date when a column was last analyzed. Database restart is not supported. Otherwise, data loss will occur.
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
notes	character varying(99)	Not supported. Its value is <b>NULL</b> .
avg_col_len	numeric	Average length of a column, in bytes.
histogram	character varying(15)	<p>Specifies whether the histogram exists and the type of the histogram if it exists.</p> <ul style="list-style-type: none"> <li><b>NONE</b>: no histogram.</li> <li><b>FREQUENCY</b>: frequency histogram.</li> <li><b>EQUI-WIDTH</b>: equal-width histogram.</li> </ul>

Name	Type	Description
scope	character varying(7)	For statistics collected on any table other than global temporary tables, the value is SHARED (indicating that the statistics are shared among all sessions).
schema	character varying(64)	Name of the namespace to which the column belongs.

### 12.3.83 DB\_TAB\_COLUMNS

DB\_TAB\_COLUMNS displays description information about columns of tables and views accessible to the current user. This view exists in both PG\_CATALOG and SYS schema. This view is accessible to all users and displays all information accessible to the current user.

**Table 12-186** DB\_TAB\_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	character varying(64)	Column name.
data_type	character varying(128)	Data type of the column.
data_type_mod	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_type_owner	character varying(128)	Owner of the data type of a column.
data_length	integer	Length of the column, in bytes.
data_precision	integer	Precision of a data type. This column is valid for the numeric data type and <b>NULL</b> for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and <b>0</b> for other data types.
nullable	bpchar	Specifies whether a column can be empty ( <b>n</b> for a primary key constraint or NOT NULL constraint).

Name	Type	Description
column_id	integer	Sequence number of the column when the table is created.
default_length	numeric	Length of the default value of a column, in bytes.
data_default	text	Default value of a column.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Minimum value in a column.
high_value	raw	Maximum value in a column.
density	numeric	Column density.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	date	Last analysis date.
sample_size	numeric	Sample size used to analyze a column.
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
char_col_decl_length	numeric	Declaration length of a column of the character type.
global_stats	character varying(3)	Not supported. Its value is <b>NO</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NO</b> .
avg_col_len	numeric	Average length of a column, in bytes.
char_length	numeric	Column length (in characters) which is valid only for the varchar, nvarchar2, bpchar, and char types.
char_used	character varying(1)	Not supported. Set it to <b>B</b> if the data type is varchar, nvarchar2, bpchar, or char, and to <b>NULL</b> for other data types.
v80_fmt_image	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_upgraded	character varying(3)	Not supported. Its value is <b>YES</b> .

Name	Type	Description
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram if it exists. <ul style="list-style-type: none"> <li>• <b>NONE</b>: no histogram.</li> <li>• <b>FREQUENCY</b>: frequency histogram.</li> <li>• <b>EQUI_WIDTH</b>: equal-width histogram.</li> </ul>
default_on_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
identity_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
evaluation_edition	character varying(128)	Not supported. Its value is <b>NULL</b> .
unusable_before	character varying(128)	Not supported. Its value is <b>NULL</b> .
unusable_beginning	character varying(128)	Not supported. Its value is <b>NULL</b> .
collation	character varying(100)	Collation rule of a column. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
comments	text	Comments.
schema	character varying(64)	Name of the namespace to which the column belongs.

### 12.3.84 DB\_TAB\_COMMENTS

DB\_TAB\_COMMENTS displays comments about all tables and views accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-187** DB\_TAB\_COMMENTS columns

Name	Type	Description
owner	character varying(128)	Owner of a table or view.
table_name	character varying(128)	Name of the table or view.
table_type	character varying(11)	Object type.
comments	text	Comments.
origin_con_id	numeric	Not supported. Its value is <b>0</b> .

Name	Type	Description
schema	character varying(64)	Name of the namespace to which the table belongs.

### 12.3.85 DB\_TAB\_HISTOGRAMS

DB\_TAB\_HISTOGRAMS displays histogram statistics about the tables or views accessible to the current user, that is, the distribution of data in each column of the table. All users can access this view. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-188** DB\_TAB\_HISTOGRAMS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(4000)	Column name.
endpoint_number	numeric	Bucket ID of the histogram.
endpoint_value	numeric	Not supported. Its value is <b>NULL</b> .
endpoint_actual_value	character varying(4000)	Actual value of the bucket endpoint.
endpoint_actual_value_raw	raw	Not supported. Its value is <b>NULL</b> .
endpoint_repeat_count	numeric	Not supported. Its value is <b>NULL</b> .
scope	character varying(7)	Not supported. Its value is <b>SHARED</b> .

### 12.3.86 DB\_TAB\_PARTITIONS

DB\_TAB\_PARTITIONS displays information about level-1 partitions (including level-2 partitioned tables) accessible to the current user. It is accessible to all users. This view exists in both PG\_CATALOG and SYS schema. The distributed system does not support level-2 partitions. Therefore, this view does not store level-1 partition information of level-2 partitioned tables.

**Table 12-189** DB\_TAB\_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.
high_value	text	Limit of a partition. <ul style="list-style-type: none"> <li>For range partitioning and interval partitioning, the upper limit of each partition is displayed.</li> <li>For list partitioning, the value list of each partition is displayed.</li> <li>For hash partitioning, the number of each partition is displayed.</li> </ul>
tablespace_name	name	Tablespace name of the partitioned table.
schema	character varying(64)	Name of a namespace.
composite	character varying(3)	Specifies whether the table is a level-2 partitioned table.
subpartition_count	numeric	Not supported. Its value is <b>NULL</b> .
high_value_length	integer	Length of the binding value expression in a partition.
partition_position	numeric	Position of the partition in the table.
pct_free	numeric	Minimum percentage of available space in a block.
pct_used	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Initial number of transactions. The default value is <b>4</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is <b>128</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Specifies whether changes to the table are logged.
compression	character varying(8)	Actual compression attribute of a partitioned table.
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_row_len	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool allocated to a partitioned block.
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
is_nested	character varying(3)	Specifies whether this partitioned table is a nested partitioned table.
parent_table_partition	character varying(128)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table. The distributed system does not support interval partitioning. Therefore, the value of this parameter is <b>NO</b> .
segment_created	character varying(4)	Specifies whether a partitioned table has segments created or has not been created.
indexing	character varying(4)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(4)	Not supported. Its value is <b>NULL</b> .
inmemory	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
inmemory_service_name	character varying(100)	Not supported. Its value is <b>NULL</b> .
memoptimize_read	character varying(8)	Not supported. Its value is <b>NULL</b> .
memoptimize_write	character varying(8)	Not supported. Its value is <b>NULL</b> .

### 12.3.87 DB\_TAB\_STATS\_HISTORY

DB\_TAB\_STATS\_HISTORY records the tables, partitions, or subpartitions involved in table statistics and the time when table statistics are collected. All users can access this view. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-190** DB\_TAB\_STATS\_HISTORY columns

Name	Type	Description
owner	character varying(128)	Object owner.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
stats_update_time	timestamp(6) with time zone	Time when statistics are updated. Database restart is not supported. Otherwise, data loss will occur.

## 12.3.88 DB\_TAB\_SUBPARTITIONS

DB\_TAB\_SUBPARTITIONS displays information about level-2 partitioned tables accessible to the current user. It is accessible to all users. This view exists in both PG\_CATALOG and SYS schema. The distributed system does not support level-2 partitioned tables. All columns in this view are set to **NULL**.

**Table 12-191** DB\_TAB\_SUBPARTITIONS columns

Name	Type	Description
table_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
table_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
high_value	text	Not supported. Its value is <b>NULL</b> .
high_value_length	numeric	Not supported. Its value is <b>NULL</b> .
partition_position	numeric	Not supported. Its value is <b>NULL</b> .
subpartition_position	numeric	Not supported. Its value is <b>NULL</b> .
tablespace_name	character varying(30)	Not supported. Its value is <b>NULL</b> .
pct_free	numeric	Not supported. Its value is <b>NULL</b> .
pct_used	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
ini_trans	numeric	Not supported. Its value is <b>NULL</b> .
max_trans	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(3)	Not supported. Its value is <b>NULL</b> .
compression	character varying(8)	Not supported. Its value is <b>NULL</b> .
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_row_len	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .
buffer_pool	character varying(7)	Not supported. Its value is <b>NULL</b> .
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Not supported. Its value is <b>NULL</b> .
segment_created	character varying(3)	Not supported. Its value is <b>NULL</b> .
indexing	character varying(3)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
inmemory	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .
cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
memoptimize_read	character varying(8)	Not supported. Its value is <b>NULL</b> .
memoptimize_write	character varying(8)	Not supported. Its value is <b>NULL</b> .

### 12.3.89 DB\_TABLES

DB\_TABLES displays all tables accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-192** DB\_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
tablespace_name	character varying(64)	Tablespace name of the table.
num_rows	numeric	Estimated number of rows in the table.

Name	Type	Description
status	character varying(8)	Specifies whether the current table is valid. <ul style="list-style-type: none"> <li>• <b>VALID</b>: The current table is valid.</li> <li>• <b>UNUSABLE</b>: The current table is unavailable.</li> </ul>
sample_size	numeric	Number of samples used for analyzing the table.
temporary	character(1)	Specifies whether the table is a temporary table. <ul style="list-style-type: none"> <li>• <b>Y</b>: The table is a temporary table.</li> <li>• <b>N</b>: The table is not a temporary table.</li> </ul>
dropped	character varying	Specifies whether the current table is deleted. <ul style="list-style-type: none"> <li>• <b>YES</b>: It is deleted.</li> <li>• <b>NO</b>: It is not deleted.</li> </ul>
pct_free	numeric	Minimum percentage of free space in a block.
ini_trans	numeric	Initial number of transactions.
max_trans	numeric	Maximum number of transactions.
avg_row_len	integer	Average number of bytes in each row.
partitioned	character varying(3)	Specifies whether a table is a partitioned table. <ul style="list-style-type: none"> <li>• <b>YES</b>: The table is a partitioned table.</li> <li>• <b>NO</b>: The table is not a partitioned table.</li> </ul>
last_analyzed	timestamp with time zone	Last time when the table was analyzed.

Name	Type	Description
row_movement	character varying(8)	Specifies whether to allow partition row movement. <ul style="list-style-type: none"> <li>● <b>ENABLED</b>: The partition row movement is allowed.</li> <li>● <b>DISABLED</b>: The partition row movement is not allowed.</li> </ul>
compression	character varying(8)	Specifies whether to enable a table compression. <ul style="list-style-type: none"> <li>● <b>ENABLED</b>: A table compression is enabled.</li> <li>● <b>DISABLED</b>: A table compression is disabled.</li> </ul>
duration	character varying(15)	Time elapsed when a temporary table is processed. <ul style="list-style-type: none"> <li>● <b>NULL</b>: The table is not a temporary table.</li> <li>● <b>sys\$session</b>: The table is a temporary session table.</li> <li>● <b>sys\$transaction</b>: The table is a temporary transaction table.</li> </ul>
logical_replication	character varying(8)	Specifies whether logical replication is enabled for a table. <ul style="list-style-type: none"> <li>● <b>ENABLED</b>: Logical replication is enabled.</li> <li>● <b>DISABLED</b>: Logical replication is disabled.</li> </ul>
external	character varying(3)	Specifies whether the table is a foreign table. <ul style="list-style-type: none"> <li>● <b>YES</b>: The table is a foreign table.</li> <li>● <b>NO</b>: The table is not a foreign table.</li> </ul>
logging	character varying(3)	Specifies whether changes to a table are logged. <ul style="list-style-type: none"> <li>● <b>YES</b>: Changes to the table are logged.</li> <li>● <b>NO</b>: Changes to the table are not logged.</li> </ul>

Name	Type	Description
default_collation	character varying(100)	Default collation of a table. <ul style="list-style-type: none"> <li>• default</li> </ul>
degree	character varying(10)	Number of instances in a scanned table.
table_lock	character varying(8)	Specifies whether to enable a table lock. <ul style="list-style-type: none"> <li>• <b>ENABLED</b>: The table lock is enabled.</li> <li>• <b>DISABLED</b>: The table lock is disabled.</li> </ul>
nested	character varying(3)	Specifies whether a table is a nested table. <ul style="list-style-type: none"> <li>• <b>YES</b>: The table is a nested table.</li> <li>• <b>NO</b>: The table is not a nested table.</li> </ul>
buffer_pool	character varying(7)	Default buffer pool of a table.
flash_cache	character varying(7)	Smart flash cache hint in database for a table block.
cell_flash_cache	character varying(7)	Cell flash cache hint for a table block.
skip_corrupt	character varying(8)	Specifies whether to skip corrupted blocks during table scanning. <ul style="list-style-type: none"> <li>• <b>ENABLED</b>: The corrupted block is skipped.</li> <li>• <b>DISABLED</b>: The corrupted block is not skipped.</li> </ul>
has_identity	character varying(3)	Specifies whether a table has an identifier column. <ul style="list-style-type: none"> <li>• <b>YES</b>: There is an identifier column.</li> <li>• <b>NO</b>: There is no identifier column.</li> </ul>
segment_created	character varying(3)	Specifies whether a table segment has been created. <ul style="list-style-type: none"> <li>• <b>YES</b>: The table segment has been created.</li> <li>• <b>NO</b>: The table segment is not created.</li> </ul>

Name	Type	Description
monitoring	character varying(3)	Specifies whether to monitor the modification of a table. <ul style="list-style-type: none"> <li>• <b>YES</b>: The modification of the table is monitored.</li> <li>• <b>NO</b>: The modification of the table is not monitored.</li> </ul>
cluster_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
iot_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
pct_used	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
backed_up	character varying(1)	Not supported. Its value is <b>NULL</b> .
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_space_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
num_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .
instances	character varying(10)	Not supported. Its value is <b>NULL</b> .
cache	character varying(5)	Not supported. Its value is <b>NULL</b> .
iot_type	character varying(12)	Not supported. Its value is <b>NULL</b> .
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
cluster_owner	character varying(30)	Not supported. Its value is <b>NULL</b> .
dependencies	character varying(8)	Not supported. Its value is <b>NULL</b> .
compression_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(3)	Not supported. Its value is <b>NULL</b> .
result_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
clustering	character varying(3)	Not supported. Its value is <b>NULL</b> .
activity_tracking	character varying(23)	Not supported. Its value is <b>NULL</b> .
dml_timestamp	character varying(25)	Not supported. Its value is <b>NULL</b> .
container_data	character varying(3)	Not supported. Its value is <b>NULL</b> .
inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
duplicated	character varying(1)	Not supported. Its value is <b>NULL</b> .
sharded	character varying(1)	Not supported. Its value is <b>NULL</b> .
hybrid	character varying(3)	Not supported. Its value is <b>NULL</b> .
cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
containers_default	character varying(3)	Not supported. Its value is <b>NULL</b> .
container_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .
container_map_object	character varying(3)	Not supported. Its value is <b>NULL</b> .
memoptimize_read	character varying(8)	Not supported. Its value is <b>NULL</b> .
memoptimize_write	character varying(8)	Not supported. Its value is <b>NULL</b> .
has_sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
admit_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_link_dml_enabled	character varying(3)	Not supported. Its value is <b>NULL</b> .
object_id_type	character varying(16)	Not supported. Its value is <b>NULL</b> .
table_type_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
table_type	character varying(128)	Not supported. Its value is <b>NULL</b> .
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .

## 12.3.90 DB\_TRIGGERS

DB\_TRIGGERS displays information about triggers accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-193** DB\_TRIGGERS columns

Name	Type	Description
trigger_name	character varying(64)	Trigger name.
table_owner	character varying(64)	Role name.
table_name	character varying(64)	Relational table name.
status	character varying(64)	<ul style="list-style-type: none"> <li>• <b>O</b>: The trigger is enabled in origin or local mode.</li> <li>• <b>D</b>: The trigger is disabled.</li> <li>• <b>R</b>: The trigger is enabled in replica mode.</li> <li>• <b>A</b>: The trigger is always enabled.</li> </ul>

## 12.3.91 DB\_TYPES

DB\_TYPES displays all object types accessible to the current user. All users can access this view. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-194** DB\_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of the type.
type_name	character varying(128)	Type name.
type_oid	raw	Type OID.
typecode	character varying(128)	Type code.
attributes	numeric	Number of attributes in a type.

Name	Type	Description
methods	numeric	Not supported. Its value is <b>0</b> .
predefined	character varying(3)	Specifies whether the type is a predefined type.
incomplete	character varying(3)	Specifies whether the type is incomplete.
final	character varying(3)	Not supported. Its value is <b>NULL</b> .
instantiable	character varying(3)	Not supported. Its value is <b>NULL</b> .
persistable	character varying(3)	Not supported. Its value is <b>NULL</b> .
supertype_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
supertype_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
local_attributes	numeric	Not supported. Its value is <b>NULL</b> .
local_methods	numeric	Not supported. Its value is <b>NULL</b> .
typeid	raw	Not supported. Its value is <b>NULL</b> .

### 12.3.92 DB\_USERS

DB\_USERS displays all users of the database visible to the current user. However, it does not describe the users. By default, only the system administrator can access this view. This view exists in both PG\_CATALOG and SYS schema.

Table 12-195 DB\_USERS columns

Name	Type	Description
user_id	oid	OID of a user.
username	name	Username.

### 12.3.93 DB\_VIEWS

DB\_VIEWS displays the description about all views accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-196** DB\_VIEWS columns

Name	Type	Description
owner	name	Owner of a view.
view_name	name	View name.
text	text	Text in the view.
text_length	integer	Text length of the view.
TEXT_VC	character varying(4000)	View creation statement. This column may truncate the view text. The BEQUEATH clause will not appear as part of the <b>TEXT_VC</b> column in this view.
type_text_length	numeric	Not supported. Its value is <b>NULL</b> .
type_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .
oid_text_length	numeric	Not supported. Its value is <b>NULL</b> .
oid_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .
view_type_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
view_type	character varying(128)	Not supported. Its value is <b>NULL</b> .
superview_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
editioning_view	character varying(1)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(1)	Not supported. Its value is <b>NULL</b> .
container_data	character varying(1)	Not supported. Its value is <b>NULL</b> .
bequeath	character varying(12)	Not supported. Its value is <b>NULL</b> .
origin_con_id	character varying(256)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
default_collation	character varying(100)	Not supported. Its value is <b>NULL</b> .
containers_default	character varying(3)	Not supported. Its value is <b>NULL</b> .
container_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
has_sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
admit_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
pdb_local_only	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.94 DICT

DICT displays the description of data dictionary tables and system views in the database. This view exists in both PG\_CATALOG and SYS schema and all users can access this view.

**Table 12-197** DICT columns

Name	Type	Description
table_name	character varying(128)	Object name.
comments	character varying(4000)	Comment on an object.

### 12.3.95 DICTIONARY

DICTIONARY displays the description of data dictionary tables and system views in the database. This view exists in both PG\_CATALOG and SYS schema and all users can access this view.

**Table 12-198** DICTIONARY columns

Name	Type	Description
table_name	character varying(128)	Object name.
comments	character varying(4000)	Comment on an object.

## 12.3.96 DV\_SESSIONS

DV\_SESSIONS displays information about all active backend threads. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

**Table 12-199** DV\_SESSIONS columns

Name	Type	Description
sid	bigint	OID of the active backend thread of the current session.
serial#	integer	Sequence number of the active backend thread, which is <b>0</b> in GaussDB.
user#	oid	OID of the user logged in to the backend thread. The OID is <b>0</b> if the backend thread is a global auxiliary thread.
username	name	Name of the user logged in to the backend thread. <b>username</b> is null if the backend thread is a global auxiliary thread. <b>application_name</b> can be identified by associating with pg_stat_get_activity(). Example: SELECT s.*,a.application_name FROM DV_SESSIONS AS s LEFT JOIN pg_stat_get_activity(NULL) AS a ON s.sid=a.sessionid;

## 12.3.97 DV\_SESSION\_LONGOPS

DV\_SESSION\_LONGOPS displays the progress of ongoing operations. The view can be accessed only after being authorized.

**Table 12-200** DV\_SESSION\_LONGOPS columns

Name	Type	Description
sid	bigint	OID of the running backend process.

Name	Type	Description
serial#	integer	Sequence number of the running backend process, which is 0 in GaussDB.
sofar	integer	Completed workload, which is null in GaussDB.
totalwork	integer	Total workload, which is null in GaussDB.

## 12.3.98 GET\_GLOBAL\_PREPARED\_XACTS

GET\_GLOBAL\_PREPARED\_XACTS displays prepared transactions on all nodes globally.

**Table 12-201** GET\_GLOBAL\_PREPARED\_XACTS columns

Name	Type	Description
transaction	xid	XID of a prepared transaction.
gid	text	GID of the prepared transaction.
prepared	timestamp with time zone	Prepared time of the prepared transaction.
owner	name	Owner of the prepared transaction.
database	name	Database to which the prepared transaction belongs.
node_name	text	Name of the node where the prepared transaction resides.

## 12.3.99 GLOBAL\_BAD\_BLOCK\_INFO

GLOBAL\_BAD\_BLOCK\_INFO is executed on the CN to collect statistics on damaged data pages of all instances. The queried result displays basic information about damaged pages. The execution result on the DN is empty. Based on the information, you can use the page detection and repair function in [Data Damage Detection and Repair Functions](#) to perform further repair operations. By default, only initial users, users with the sysadmin permission, users with the O&M administrator permission in the O&M mode, and monitoring users can view the information. Other users can view the information only after being granted with permissions.

**Table 12-202** GLOBAL\_BAD\_BLOCK\_INFO columns

Name	Type	Description
node_name	text	Information about the node where the current damaged page is located.
spc_node	oid	ID of the tablespace corresponding to the current damaged page.
db_node	oid	ID of the database corresponding to the current damaged page.
rel_node	oid	Relfilenode of the relation corresponding to the current damaged page.
bucket_node	integer	Bucket node of the current damaged page. It is set to <b>-1</b> for a segment-page table. Other values are reserved and not supported.
block_num	oid	Page number of the current damaged page.
fork_num	integer	File forknum of the current damaged page.
file_path	text	Relative path of the current damaged page.
check_time	timestamp with time zone	Time when an error is detected on the current damaged page.
repair_time	timestamp with time zone	Time when the current damaged page is repaired.

### 12.3.100 GLOBAL\_CLEAR\_BAD\_BLOCK\_INFO

GLOBAL\_CLEAR\_BAD\_BLOCK\_INFO is executed on the CN to clear information about repaired pages in all instances. The execution result on DNs is empty. By default, only initial users, users with the sysadmin permission, users with the O&M administrator permission in the O&M mode, and monitoring users can view the information. Other users can view the information only after being granted with permissions.

**Table 12-203** GLOBAL\_BAD\_BLOCK\_INFO columns

Name	Type	Description
node_name	text	Information about the node for clearing repaired pages.
result	Boolean	Execution result of clearing repaired pages of the current instance

## 12.3.101 GLOBAL\_COMM\_CLIENT\_INFO

GLOBAL\_COMM\_CLIENT\_INFO queries information about active client connections of global nodes in a cluster. By default, only the system administrator has the permission to access this system view.

**Table 12-204** GLOBAL\_COMM\_CLIENT\_INFO columns

Name	Type	Description
node_name	text	Current node name.
app	text	App
tid	bigint	Thread ID of the current thread.
lwtid	integer	Lightweight thread ID of the current thread.
query_id	bigint	Query ID. It is equivalent to <b>debug_query_id</b> .
socket	integer	Displayed if the connection is a physical connection.
remote_ip	text	Peer node IP address.
remote_port	text	Peer node port.
logic_id	integer	Displayed if the connection is a logical connection.

## 12.3.102 GLOBAL\_SQL\_PATCH

GLOBAL\_SQL\_PATCH displays information about all SQL patches. This view is available only in the PG\_CATALOG schema.

**Table 12-205** GLOBAL\_SQL\_PATCH columns

Name	Type	Description
node_name	text	Name of the node where the SQL PATCH is located.
patch_name	name	Name of the SQL PATCH.
unique_sql_id	bigint	Global unique query ID.
owner	oid	ID of the user created using the SQL PATCH.
enable	Boolean	Specifies whether the SQL patch takes effect.
status	"char"	SQL patch status (reserved column).

Name	Type	Description
abort	Boolean	Specifies whether a hint is discarded.
hint_string	text	Hint text.
description	text	SQL patch description.
parent_unique_sql_id	bigint	<p>Globally unique ID of the outer statement of the SQL statement for which SQL patch takes effect.</p> <ul style="list-style-type: none"> <li>• Its value is <b>0</b> for statements outside the stored procedure.</li> <li>• For a statement within a stored procedure, the value is the globally unique ID of the statement that calls the stored procedure.</li> </ul>

### 12.3.103 GLOBAL\_STAT\_HOTKEYS\_INFO

GLOBAL\_STAT\_HOTKEYS\_INFO is used to query the statistics of hotspot keys in the entire cluster. The query results are sorted by **count** in descending order.

**Table 12-206** GLOBAL\_STAT\_HOTKEYS\_INFO columns

Name	Type	Description
database_name	text	Name of the database where the hot key is located.
schema_name	text	Name of the mode where the hotspot key is located.
table_name	text	Name of the table where the hotspot key is located.
key_value	text	Value of the hotspot key.
hash_value	bigint	Hash value of the hotspot key in the database. If the table is a list or range distributed table, the value of this column is <b>0</b> .
count	numeric	Frequency of accessing the hotspot key.

## 12.3.104 GLOBAL\_WAL\_SENDER\_STATUS

GLOBAL\_WAL\_SENDER\_STATUS displays the redo log transfer and replay status of the primary DN in the current cluster. This view can be viewed only by the users with monitor admin and sysadmin permission.

**Table 12-207** GLOBAL\_WAL\_SENDER\_STATUS column

Name	Type	Description
nodename	text	Name of the primary node.
source_ip	text	IP address of the primary node.
source_port	integer	Port of the primary node.
dest_ip	text	IP address of the standby node.
dest_port	integer	Port of the standby node.
sender_pid	integer	PID of the sending thread.
local_role	text	Type of the primary node. <ul style="list-style-type: none"> <li>• <b>UNKNOWN_MODE</b>: The status is unknown.</li> <li>• <b>NORMAL_MODE</b>: single-host node.</li> <li>• <b>PRIMARY_MODE</b>: primary node.</li> <li>• <b>STANDBY_MODE</b>: standby node.</li> <li>• <b>CASCADE_STANDBY_MODE</b>: cascaded standby node.</li> <li>• <b>PENDING_MODE</b>: The node is in the arbitration phase.</li> <li>• <b>RECOVERY_MODE</b>: The node is in the restoration phase.</li> <li>• <b>STANDBY_CLUSTER_MODE</b>: standby cluster node.</li> <li>• <b>MAIN_STANDBY_MODE</b>: main standby cluster node.</li> </ul> <p><b>NOTE</b> The expected values of the primary node type are <b>NORMAL_MODE</b>, <b>PRIMARY_MODE</b>, <b>PENDING_MODE</b>, and <b>RECOVERY_MODE</b>. If other node types are displayed, contact Huawei technical support.</p>

Name	Type	Description
peer_role	text	<p>Type of the standby node.</p> <ul style="list-style-type: none"> <li>● <b>UNKNOWN_MODE</b>: The status is unknown.</li> <li>● <b>NORMAL_MODE</b>: single-host node.</li> <li>● <b>PRIMARY_MODE</b>: primary node.</li> <li>● <b>STANDBY_MODE</b>: standby node.</li> <li>● <b>CASCADE_STANDBY_MODE</b>: cascaded standby node.</li> <li>● <b>PENDING_MODE</b>: The node is in the arbitration phase.</li> <li>● <b>RECOVERY_MODE</b>: The node is in the restoration phase.</li> <li>● <b>STANDBY_CLUSTER_MODE</b>: standby cluster node.</li> <li>● <b>MAIN_STANDBY_MODE</b>: main standby cluster node.</li> </ul> <p><b>NOTE</b> The expected values of the standby node type are <b>NORMAL_MODE</b>, <b>STANDBY_MODE</b>, <b>CASCADE_STANDBY_MODE</b>, <b>PENDING_MODE</b>, <b>RECOVERY_MODE</b>, <b>STANDBY_CLUSTER_MODE</b>, and <b>MAIN_STANDBY_MODE</b>. If other node types are displayed, contact Huawei technical support.</p>

Name	Type	Description
peer_state	text	<p>Status of the standby node.</p> <ul style="list-style-type: none"> <li>• <b>UNKNOWN_STATE</b>: The node status is unknown.</li> <li>• <b>NORMAL_STATE</b>: The node is started properly.</li> <li>• <b>NEEDREPAIR_STATE</b>: The current node needs to be repaired.</li> <li>• <b>STARTING_STATE</b>: The node is being started.</li> <li>• <b>WAITING_STATE</b>: The node is waiting to be upgraded.</li> <li>• <b>DEMOTING_STATE</b>: The node is being downgraded.</li> <li>• <b>PROMOTING_STATE</b>: The standby node is being upgraded to the primary node.</li> <li>• <b>BUILDING_STATE</b>: The standby node fails to be started and needs to be rebuilt.</li> <li>• <b>CATCHUP_STATE</b>: The standby node is catching up with the primary node.</li> <li>• <b>COREDUMP_STATE</b>: The node breaks down.</li> </ul>
state	text	<p>WAL sender status.</p> <ul style="list-style-type: none"> <li>• <b>WALSNDSTATE_STARTUP</b>: startup state.</li> <li>• <b>WALSNDSTATE_BACKUP</b>: backup state.</li> <li>• <b>WALSNDSTATE_CATCHUP</b>: catch-up state.</li> <li>• <b>WALSNDSTATE_STREAMING</b>: streaming replication state.</li> </ul>
sender_sent_location	text	Sending position of the primary node.
sender_write_location	text	Writing position of the primary node.
sender_flush_location	text	Flushing position of the primary node.
sender_replay_location	text	Redo position of the primary node.
receiver_received_location	text	Receiving position of the standby node.
receiver_write_location	text	Writing position of the standby node.

Name	Type	Description
receiver_flush_location	text	Flushing location of the standby node.
receiver_replay_location	text	Redo location of the standby node.

## 12.3.105 GS\_ALL\_CONTROL\_GROUP\_INFO

GS\_ALL\_CONTROL\_GROUP\_INFO displays all Cgroup information in a database.

**Table 12-208** GS\_ALL\_CONTROL\_GROUP\_INFO columns

Name	Type	Description
name	text	Name of a Cgroup.
type	text	Type of the Cgroup. <ul style="list-style-type: none"> <li>● <b>GROUP_NONE</b>: no group.</li> <li>● <b>GROUP_TOP</b>: top group.</li> <li>● <b>GROUP_CLASS</b>: class Cgroup, which does not control any thread.</li> <li>● <b>GROUP_BAKWD</b>: backend thread Cgroup.</li> <li>● <b>GROUP_DEFWD</b>: default Cgroup, which controls only the query threads at this level.</li> <li>● <b>GROUP_TSWD</b>: time-sharing Cgroup of each user, which controls the query threads at the bottom layer.</li> </ul>
gid	bigint	Cgroup ID.
classgid	bigint	ID of the class Cgroup where a workload Cgroup belongs.
class	text	Class Cgroup.
workload	text	Workload Cgroup.
shares	bigint	CPU quota allocated to the Cgroup.
limits	bigint	Limit of CPUs allocated to the Cgroup.
wdlevel	bigint	<b>Workload</b> Cgroup level.
cpucores	text	Information about the CPU cores used by a Cgroup.

## 12.3.106 GS\_ALL\_PREPARED\_STATEMENTS

GS\_ALL\_PREPARED\_STATEMENTS displays information about prepared statements that are available in all sessions. Only the system administrator can access this view.

**Table 12-209** GS\_ALL\_PREPARED\_STATEMENTS columns

Name	Type	Description
pid	bigint	Backend thread ID. <b>NOTE</b> In thread pool mode, <b>pid</b> indicates the ID of the thread bound to the current session. When a session is executed on different threads, <b>pid</b> changes accordingly. In thread pool mode, statements are associated with <b>sessionid</b> but not <b>pid</b> . You are advised to use <b>sessionid</b> for join query.
sessionid	bigint	Current session ID.
global_sessionid	text	Global session ID.
name	text	Identifier of a prepared statement.
statement	text	Query string for creating this prepared statement. For prepared statements created through SQL, this is the PREPARE statement committed by the client. For prepared statements created through the frontend/backend protocol, this is the text of the prepared statement itself.
prepare_time	timestamp with time zone	Timestamp when the prepared statement is created.
parameter_types	regtype[]	Expected parameter types for the prepared statement in the form of an array of <b>regtype</b> . The OID corresponding to an element of this array can be obtained by converting the <b>regtype</b> value to an OID value.
from_sql	Boolean	<ul style="list-style-type: none"><li>Its value is <b>true</b> if the prepared statement was created through the PREPARE statement</li><li>Its value is <b>false</b> if the statement was prepared through the frontend/backend protocol</li></ul>

## 12.3.107 GS\_AUDITING

GS\_AUDITING displays all audit information about database-related operations. Only the users with system administrator or security policy administrator permission can access this view.

**Table 12-210** GS\_AUDITING columns

Name	Type	Description
polname	name	Policy name, which must be unique.
pol_type	text	Audit policy type. The value can be 'access' or 'privilege'. <ul style="list-style-type: none"> <li>● <b>access</b>: DML operations are audited.</li> <li>● <b>privilege</b>: DDL operations are audited.</li> </ul>
polenabled	Boolean	Specifies whether to enable the policy. <ul style="list-style-type: none"> <li>● <b>t</b> (true): enabled.</li> <li>● <b>f</b> (false): disabled.</li> </ul>
access_type	name	DML database operation type. For example, SELECT, INSERT, and DELETE.
label_name	name	Resource label name. This column corresponds to the <b>polname</b> column in the GS_AUDITING_POLICY system catalog.
priv_object	text	Describes the path of the database asset.
filter_name	text	Logical character string of a filter criterion.

## 12.3.108 GS\_AUDITING\_ACCESS

GS\_AUDITING\_ACCESS displays all audit information about database DML-related operations. Only the users with system administrator or security policy administrator permission can access this view.

**Table 12-211** GS\_AUDITING\_ACCESS columns

Name	Type	Description
polname	name	Policy name, which must be unique.
pol_type	text	Audit policy type. The value 'access' indicates that DML operations are audited.
polenabled	Boolean	Specifies whether to enable the policy. <ul style="list-style-type: none"> <li>● <b>t</b> (true): enabled.</li> <li>● <b>f</b> (false): disabled.</li> </ul>

Name	Type	Description
access_type	name	DML database operation type. For example, SELECT, INSERT, and DELETE.
label_name	name	Resource label name. This column corresponds to the <b>polname</b> column in the GS_AUDITING_POLICY system catalog.
access_object	text	Describes the path of the database asset.
filter_name	text	Logical character string of a filter criterion.

### 12.3.109 GS\_AUDITING\_PRIVILEGE

GS\_AUDITING\_PRIVILEGE displays all audit information about database DDL-related operations. Only the users with system administrator or security policy administrator permission can access this view.

**Table 12-212** GS\_AUDITING\_PRIVILEGE columns

Name	Type	Description
polname	name	Policy name, which must be unique.
pol_type	text	Audit policy type. The value ' <b>privilege</b> ' indicates that DDL operations are audited.
polenabled	Boolean	Specifies whether to enable the policy. <ul style="list-style-type: none"><li>• <b>t</b> (true): enabled.</li><li>• <b>f</b> (false): disabled.</li></ul>
access_type	name	DDL database operation type. For example, CREATE, ALTER, and DROP.
label_name	name	Resource label name. This column corresponds to the <b>polname</b> column in the GS_AUDITING_POLICY system catalog.
priv_object	text	Full domain name of a database object.
filter_name	text	Logical character string of a filter criterion.

### 12.3.110 GS\_CLUSTER\_RESOURCE\_INFO

GS\_CLUSTER\_RESOURCE\_INFO displays all DNs resource summaries. This view can be queried only when **enable\_dynamic\_workload** is set to **on** and the view cannot be executed on DNs. Only the user with sysadmin permission can query this view.

**Table 12-213** GS\_CLUSTER\_RESOURCE\_INFO columns

Name	Type	Description
min_mem_util	integer	Minimum memory usage of a DN
max_mem_util	integer	Maximum memory usage of the DN
min_cpu_util	integer	Minimum CPU usage of the DN
max_cpu_util	integer	Maximum CPU usage of the DN
min_io_util	integer	Minimum I/O usage of the DN
max_io_util	integer	Maximum I/O usage of the DN
used_mem_rate	integer	Maximum physical memory usage

### 12.3.111 GS\_COMM\_LISTEN\_ADDRESS\_EXT\_INFO

GS\_COMM\_LISTEN\_ADDRESS\_EXT\_INFO queries DFX information, such as threads, sessions, and socket, for connecting the extended IP addresses of the current DN. For details about the application scenarios, see

•[gs\\_comm\\_listen\\_address\\_ext\\_info](#).

**Table 12-214** GS\_COMM\_LISTEN\_ADDRESS\_EXT\_INFO columns

Name	Type	Description
node_name	text	Name of the current instance.
app	text	Client connected to the DN.
tid	bigint	Thread ID of the current thread.
lwtid	integer	Lightweight thread ID of the current thread.
query_id	bigint	Query ID of the current thread.
socket	integer	Socket FD of the current physical connection.
remote_ip	text	Peer IP address of the current connection.
remote_port	text	Peer port of the current connection.
local_ip	text	Local IP address of the current connection.
local_port	text	Local port of the current connection.

## 12.3.112 GS\_DB\_LINKS

GS\_DB\_LINKS displays information about database links. You can view information about database links of your own and at the PUBLIC level.

**Table 12-215** GS\_DB\_LINKS columns

Name	Type	Description
dblinkid	oid	OID of the current database link.
dlname	name	Name of the current database link.
downer	oid	ID of the owner of the current database link. If the owner is <b>public</b> , the value is <b>0</b> .
downername	name	Name of the owner of the current database link.
options	text[ ]	Connection information of the current database link. The value is a character string in the "keyword=value" format.
useroptions	text	User information used by the remote end of the current database link.
heterogeneous	text	Not supported. Its value is <b>NULL</b> .
protocol	text	Not supported. Its value is <b>NULL</b> .
opencursors	text	Not supported. Its value is <b>NULL</b> .
intransaction	Boolean	Determines whether the current database link exists in a transaction.
updatesent	Boolean	Determines whether the current database link uses statements for updating data.

## 12.3.113 GS\_DB\_PRIVILEGES

GS\_DB\_PRIVILEGES displays the granting of ANY permissions. Each record corresponds to a piece of authorization information.

**Table 12-216** GS\_DB\_PRIVILEGES columns

Name	Type	Description
rolename	name	Username
privilege_type	text	ANY permission of a user. For details about the value, see <a href="#">Table 7-166</a> .

Name	Type	Description
admin_option	text	Whether the ANY permission recorded in the <b>privilege_type</b> column can be re-granted <ul style="list-style-type: none"> <li>• <b>yes</b></li> <li>• <b>no</b></li> </ul>

### 12.3.114 GS\_GET\_CONTROL\_GROUP\_INFO

GS\_GET\_CONTROL\_GROUP\_INFO displays information about all Cgroups. Only the user with sysadmin permission can query this view.

Table 12-217 GS\_GET\_CONTROL\_GROUP\_INFO columns

Name	Type	Description
group_name	text	Name of a Cgroup.
group_type	text	Type of the Cgroup. <ul style="list-style-type: none"> <li>• <b>GROUP_NONE</b>: no group.</li> <li>• <b>GROUP_TOP</b>: top group.</li> <li>• <b>GROUP_CLASS</b>: class Cgroup, which does not control any thread.</li> <li>• <b>GROUP_BAKWD</b>: backend thread Cgroup.</li> <li>• <b>GROUP_DEFWD</b>: default Cgroup, which controls only the query threads at this level.</li> <li>• <b>GROUP_TSWD</b>: time-sharing Cgroup of each user, which controls the query threads at the bottom layer.</li> </ul>
gid	bigint	Cgroup ID.
classgid	bigint	ID of the class Cgroup where a workload Cgroup belongs.
class	text	Class Cgroup.
group_workload	text	Workload Cgroup.
shares	bigint	CPU quota allocated to the Cgroup.

Name	Type	Description
limits	bigint	Limit of CPUs allocated to the Cgroup.
wdlevel	bigint	Workload Cgroup level.
cpucoros	text	Information about the CPU cores used by a Cgroup.
nodegroup	text	Node group name.
group_kind	text	Node group type. The value must be <b>i</b> , <b>n</b> , <b>v</b> , or <b>e</b> . <ul style="list-style-type: none"><li>• <b>i</b>: installation node group.</li><li>• <b>n</b>: node group in a common cluster.</li><li>• <b>e</b>: elastic cluster.</li></ul>

### 12.3.115 GS\_GET\_LISTEN\_ADDRESS\_EXT\_INFO

GS\_GET\_LISTEN\_ADDRESS\_EXT\_INFO queries the extended IP address configured for the current DN instance. For details about the application scenarios, see [gs\\_get\\_listen\\_address\\_ext\\_info\(\)](#).

**Table 12-218** GS\_GET\_LISTEN\_ADDRESS\_EXT\_INFO columns

Name	Type	Description
node_name	text	DN name.
host	text	Listening IP address of a DN.
port	text	Listening port of a DN.
ext_listen_ip	text	Extended IP address configured for a DN.

### 12.3.116 GS\_GLOBAL\_ARCHIVE\_STATUS

GS\_GLOBAL\_ARCHIVE\_STATUS describes the archiving progress of CNs and all shards, including the shard name (**node\_name**), archiving location (**restart\_lsn**), name of the primary or standby node (**archive\_node**), and current log location (**current\_xlog\_location**). To query this view, you need to enable the archiving function of the database and query the view from the CN node. The monitor admin and sysadmin permissions are required.

**Table 12-219** GS\_GLOBAL\_ARCHIVE\_STATUS columns

Name	Type	Description
node_name	text	Shard name.
restart_lsn	text	Archiving location.
archive_node	text	Name of the primary or standby node where archiving is performed.
current_xlog_location	text	Current log position.

### 12.3.117 GS\_GSC\_MEMORY\_DETAIL

GS\_GSC\_MEMORY\_DETAIL displays the global SysCache memory usage of the current process on the current node. The data is displayed only when Global SysCache is enabled.

Note that the query is separated by the database memory context. Therefore, some memory statistics are missing. The memory context corresponding to the missing memory statistics is **GlobalSysDBCACHE**.

**Table 12-220** GS\_GSC\_MEMORY\_DETAIL columns

Name	Type	Description
db_id	text	Database ID.
totalsize	numeric	Total size of the shared memory, in bytes.
freesize	numeric	Remaining size of the shared memory, in bytes.
usedsize	numeric	Used size of the shared memory, in bytes.

### 12.3.118 GS\_LABELS

GS\_LABELS displays all configured resource labels. Only the users with system administrator or security policy administrator permission can access this view.

**Table 12-221** GS\_LABELS columns

Name	Type	Description
labelname	name	Resource label name.
labeltype	name	Resource label type This parameter corresponds to the <b>labeltype</b> column in the <a href="#">GS_POLICY_LABEL</a> system catalog.

Name	Type	Description
fqdtype	name	Database resource type. For example, table, schema, and index.
schemaname	name	Name of the schema to which the database resource belongs.
fqdnname	name	Database resource name.
columnname	name	Name of the database resource column. If the marked database resource is not a column, this parameter is left blank.

### 12.3.119 GS\_LSC\_MEMORY\_DETAIL

GS\_LSC\_MEMORY\_DETAIL displays the memory usage of the local system cache of all threads based on the MemoryContext node. The data is displayed only when Global SysCache is enabled.

**Table 12-222** GS\_LSC\_MEMORY\_DETAIL columns

Name	Type	Description
threadid	text	Thread start time and ID (string: <i>timestamp.sessionid</i> ).
tid	bigint	Thread ID.
thrdtype	text	Thread type. It can be any thread type in the system, such as postgresql and wlmmonitor.
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.
parent	text	Name of the parent memory context.
totalsize	bigint	Total size of the memory context, in bytes.
freesize	bigint	Total size of released memory in the memory context, in bytes.
usedsize	bigint	Total size of used memory in the memory context, in bytes.

### 12.3.120 GS\_MASKING

GS\_MASKING displays all configured dynamic masking policies. Only the users with system administrator or security policy administrator permission can access this view.

**Table 12-223** GS\_MASKING columns

Name	Type	Description
polname	name	Name of the masking policy
polenabed	Boolean	Specifies whether to enable the masking policy.
maskaction	name	Masking function
labelname	name	Name of the label to which the masking function applies.
masking_object	text	Masking database resource object
filter_name	text	Logical expression of a filter criterion

### 12.3.121 GS\_MATVIEWS

**GS\_MATVIEWS** provides information about each materialized view in the database.

**Table 12-224** GS\_MATVIEWS columns

Name	Type	Reference	Description
schemaname	name	<a href="#">PG_NAMESPACE</a> .nspname	Name of the schema of a materialized view.
matviewname	name	<a href="#">PG_CLASS</a> .relname	Name of a materialized view.
matviewowner	name	<a href="#">PG_AUTHID</a> .Erolname	Owner of a materialized view.
tablespace	name	<a href="#">PG_TABLESPACE</a> .spcname	Tablespace name of a materialized view. If the default tablespace of the database is used, the value is null.
hasindexes	boolean	-	This column is true if a materialized view has (or has recently had) any indexes.
definition	text	-	Definition of a materialized view (a reconstructed SELECT query).

### 12.3.122 GS\_MY\_PLAN\_TRACE

**GS\_MY\_PLAN\_TRACE** is a view of the **GS\_PLAN\_TRACE** system catalog. This view displays the plan traces of the current user. The plan trace feature is not supported

in the distributed scenario. Therefore, no data is displayed in this view in the distributed scenario.

**Table 12-225** GS\_MY\_PLAN\_TRACE columns

Name	Type	Description
query_id	text	Unique ID of the current request
query	text	SQL statement of the current request. The value of this column cannot exceed the value of <b>track_activity_query_size</b> .
unique_sql_id	bigint	Unique ID of the SQL statement of the current request
plan	text	Query plan text corresponding to the SQL statement of the current request. The size of this column cannot exceed 10 KB.
plan_trace	text	Details about the query plan generation process corresponding to the SQL statement of the current request. The value of this column cannot exceed 300 MB.
modifydate	timestamp with time zone	Time when the current plan trace is updated (that is, time when the plan trace is created)

### 12.3.123 GS\_SESSION\_ALL\_SETTINGS

GS\_SESSION\_ALL\_SETTINGS displays the full GUC parameter settings of all sessions on the local node. Only users with the sysadmin or monadmin permissions can query this view.

**Table 12-226** GS\_SESSION\_ALL\_SETTINGS columns

Name	Type	Description
sessionid	bigint	Session ID.
pid	bigint	Backend thread ID.
name	text	Parameter name.
setting	text	Current parameter value.
unit	text	Implicit unit of a parameter.

### 12.3.124 GS\_SQL\_COUNT

GS\_SQL\_COUNT displays statistics about five types of running statements (SELECT, INSERT, UPDATE, DELETE, and MERGE INTO) on the current node of the database.

- When a common user queries the GS\_SQL\_COUNT view, statistics about the current node of the user are displayed. When an administrator queries the GS\_SQL\_COUNT view, statistics about the current node of all users are displayed.
- When the cluster or node is restarted, the statistics are cleared and will be measured again.
- The system counts when a node receives a query, including a query inside the cluster. For example, when a CN receives a query and distributes multiple queries to DNs, the queries are counted accordingly on the DNs.

**Table 12-227** GS\_SQL\_COUNT columns

Name	Type	Description
node_name	name	Node name.
user_name	name	Username.
select_count	bigint	Statistical result of the SELECT statement.
update_count	bigint	Statistical result of the UPDATE statement.
insert_count	bigint	Statistical result of the INSERT statement.
delete_count	bigint	Statistical result of the DELETE statement.
mergeinto_count	bigint	Statistical result of the MERGE INTO statement.
ddl_count	bigint	Number of DDL statements.
dml_count	bigint	Number of DML statements.
dcl_count	bigint	Number of DCL statements.
total_select_elapse	bigint	Total response time of SELECT statements (unit: $\mu$ s).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: $\mu$ s).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: $\mu$ s).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: $\mu$ s).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: $\mu$ s).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: $\mu$ s).

Name	Type	Description
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: $\mu$ s).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: $\mu$ s).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: $\mu$ s).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: $\mu$ s).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: $\mu$ s).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: $\mu$ s).
total_delete_elapse	bigint	Total response time of DELETE statements (unit: $\mu$ s).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: $\mu$ s).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: $\mu$ s).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: $\mu$ s).

### 12.3.125 GS\_STAT\_ALL\_PARTITIONS

GS\_STAT\_ALL\_PARTITIONS contains information about each partition in all partitioned tables in the current database. Each partition occupies one row, showing the access statistics of the partition. The information can be queried by using the `gs_stat_get_all_partitions_stats()` function.

**Table 12-228** GS\_STAT\_ALL\_PARTITIONS columns

Name	Type	Description
partition_oid	oid	Partition OID.
schemaname	name	Schema name of a table where the partition is located.
relname	name	Name of the table where the partition is located.
partition_name	name	Name of the level-1 partition to which the partition belongs.

Name	Type	Description
sub_partition_name	name	Name of the level-2 partition to which the partition belongs. The distributed system does not support level-2 partitions. Set it to <b>NULL</b> .
seq_scan	bigint	Number of sequential scans initiated by the partition.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated by the partition.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Time when the partition was last cleared.
last_autovacuum	timestamp with time zone	Time when the partition was last cleared by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Time when the partition was last analyzed.
last_autoanalyze	timestamp with time zone	Time when the partition was last analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the partition is cleared.
autovacuum_count	bigint	Number of times that the partition is cleared by the autovacuum daemon thread.
analyze_count	bigint	Number of times that the partition is analyzed.
autoanalyze_count	bigint	Number of times that the partition is analyzed by the autovacuum daemon thread.

## 12.3.126 GS\_STAT\_XACT\_ALL\_PARTITIONS

GS\_STAT\_XACT\_ALL\_PARTITIONS displays the transaction status about all partitions of partitioned tables in a namespace. The information can be queried by using the `gs_stat_get_xact_all_partitions_stats()` function.

**Table 12-229** GS\_STAT\_XACT\_ALL\_PARTITIONS columns

Name	Type	Description
partition_oid	oid	Partition OID.
schemaname	name	Schema name of the partition.
relname	name	Name of the table where the partition is located.
partition_name	name	Name of the level-1 partition to which the partition belongs.
sub_partition_name	name	Name of the level-2 partition to which the partition belongs. The distributed system does not support level-2 partitions. Set it to <b>NULL</b> .
seq_scan	bigint	Number of sequential scans initiated by the partition.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated by the partition.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required).

## 12.3.127 GS\_STATIO\_ALL\_PARTITIONS

GS\_STATIO\_ALL\_PARTITIONS contains I/O statistics about each partition in a partitioned table of the current database. The information can be queried by using the `gs_statio_get_all_partitions_stats()` function.

**Table 12-230** GS\_STATIO\_ALL\_PARTITIONS columns

Name	Type	Description
partition_oid	oid	Partition OID.

Name	Type	Description
schemaname	name	Name of the schema that the partition is in.
relname	name	Name of the table where the partition is located.
partition_name	name	Name of the level-1 partition to which the partition belongs.
sub_partition_name	name	Name of the level-2 partition to which the partition belongs. The distributed system does not support level-2 partitions. Set it to <b>NULL</b> .
heap_blks_read	bigint	Number of disk blocks read from the partition.
heap_blks_hit	bigint	Number of cache hits in the partition.
idx_blks_read	bigint	Number of disk blocks read from all indexes on the partition.
idx_blks_hit	bigint	Number of cache hits of all indexes in a partition.
toast_blks_read	bigint	Number of disk blocks read from the partition's TOAST table partition (if any).
toast_blks_hit	bigint	Number of buffer hits in the partition's TOAST table partition (if any).
tidx_blks_read	bigint	Number of disk blocks read from the partition's TOAST table partitioned indexes (if any).
tidx_blks_hit	bigint	Number of buffer hits in the partition's TOAST table partitioned indexes (if any).

### 12.3.128 GS\_WORKLOAD\_RULE\_STAT

GS\_WORKLOAD\_RULE\_STAT displays information about SQL concurrency control rules. Only the sysadmin user can access the system view.

**Table 12-231** GS\_WORKLOAD\_RULE\_STAT columns

Name	Type	Description
rule_id	bigint	Concurrency control rule ID.
rule_name	name	Name of a concurrency control rule, which is used to search for the concurrency control rule.

Name	Type	Description
databases	name[]	List of databases on which the concurrency control rules take effect. If the value is <b>NULL</b> , the concurrency control rules take effect for all databases.
rule_type	text	Concurrency control rule type. Currently, only " <b>sqlid</b> ", " <b>select</b> ", " <b>insert</b> ", " <b>update</b> ", " <b>delete</b> ", " <b>merge</b> ", and " <b>resource</b> " are supported. Other values are invalid.
start_time	timestamp with time zone	Start time of the concurrency control rules. The value <b>NULL</b> indicates that the rules take effect from now on.
end_time	timestamp with time zone	End time of the concurrency control rules. The value <b>NULL</b> indicates that the rules are always effective.
max_workload	bigint	Maximum number of concurrent rule settings.
option_val	text[]	Parameter values of a concurrency control rule, including SQL ID, keyword list, and resource restriction.  For details, see the description of the <a href="#">gs_add_workload_rule</a> API.
is_valid	Boolean	Determines whether the concurrency control rules take effect. If the concurrency control rules time out, the value is set to <b>false</b> .
validate_count	bigint	Number of SQL statements intercepted by the concurrency control rule.
node_names	text[]	List of nodes on which the concurrency control rules take effect. This parameter is reserved and does not take effect currently.
user_names	text[]	List of users for which the concurrency control rules take effect. This parameter is reserved and does not take effect currently.

## 12.3.129 GV\_INSTANCE

GV\_INSTANCE displays information about the current database instance. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-232** GV\_INSTANCE columns

Name	Type	Description
inst_id	oid	OID of the current database.
instance_number	oid	OID of the current database.
instance_name	character varying(16)	Name of the current database.
host_name	character varying(64)	Not supported. Its value is <b>NULL</b> .
version	character varying(17)	Not supported. Its value is <b>NULL</b> .
version_legacy	character varying(17)	Not supported. Its value is <b>NULL</b> .
version_full	character varying(17)	Not supported. Its value is <b>NULL</b> .
startup_time	date	Not supported. Its value is <b>NULL</b> .
status	character varying(12)	Not supported. Its value is <b>NULL</b> .
parallel	character varying(3)	Not supported. Its value is <b>NULL</b> .
thread#	numeric	Not supported. Its value is <b>NULL</b> .
archiver	character varying(7)	Not supported. Its value is <b>NULL</b> .
log_switch_wait	character varying(15)	Not supported. Its value is <b>NULL</b> .
logins	character varying(10)	Not supported. Its value is <b>NULL</b> .
shutdown_pending	character varying(3)	Not supported. Its value is <b>NULL</b> .
database_status	character varying(17)	Not supported. Its value is <b>NULL</b> .
instance_role	character varying(18)	Not supported. Its value is <b>NULL</b> .
active_state	character varying(9)	Not supported. Its value is <b>NULL</b> .
blocked	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
con_id	numeric	Not supported. Its value is <b>NULL</b> .
instance_mode	character varying(11)	Not supported. Its value is <b>NULL</b> .
edition	character varying(7)	Not supported. Its value is <b>NULL</b> .
family	character varying(80)	Not supported. Its value is <b>NULL</b> .
database_type	character varying(15)	Not supported. Its value is <b>NULL</b> .

### 12.3.130 GV\_SESSION

GV\_SESSION describes information about all current sessions. Only administrators can access this view. Common users can access this view only after being authorized. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-233** GV\_SESSION columns

Name	Type	Description
inst_id	numeric	Not supported. Its value is <b>NULL</b> .
saddr	raw	Not supported. Its value is <b>NULL</b> .
sid	bigint	Session ID.
serial#	integer	Sequence number of the active backend thread, which is <b>0</b> in GaussDB.
audsid	numeric	Not supported. Its value is <b>NULL</b> .
paddr	raw	Not supported. Its value is <b>NULL</b> .
schema#	numeric	Not supported. Its value is <b>NULL</b> .
schemaname	name	Name of the user logged in to the backend.
user#	oid	OID of the user who has logged in to the backend thread. Its value is <b>0</b> if the backend thread is a global auxiliary thread.
username	name	Name of the user logged in to the backend thread. <b>username</b> is null if the backend thread is a global auxiliary thread.
command	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
ownerid	numeric	Not supported. Its value is <b>NULL</b> .
taddr	character varying(16)	Not supported. Its value is <b>NULL</b> .
lockwait	character varying(16)	Not supported. Its value is <b>NULL</b> .
machine	text	Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.
sql_id	bigint	ID of a query.
client_info	text	Client information.
event	text	Queuing status of a statement. Its value can be: <ul style="list-style-type: none"> <li>• <b>waiting in queue</b>: The statement is in the queue.</li> <li>• <b>Empty</b>: The statement is running.</li> </ul>
sql_exec_start	timestamp with time zone	Time when the currently active query was started, or if <b>state</b> is not <b>active</b> , when the last query was started.
program	text	Name of the application connected to the backend.
status	text	Overall status of this backend. Its value can be: <ul style="list-style-type: none"> <li>• <b>active</b>: The backend is executing a query.</li> <li>• <b>idle</b>: The backend is waiting for a new client command.</li> <li>• <b>idle in transaction</b>: The backend is in a transaction, but there is no statement being executed in the transaction.</li> <li>• <b>idle in transaction (aborted)</b>: The backend is in a transaction, but there are statements failed in the transaction.</li> <li>• <b>fastpath function call</b>: The backend is executing a fast-path function.</li> <li>• <b>disabled</b>: This state is reported if <b>track_activities</b> is disabled in this backend.</li> </ul>
server	character varying(9)	Not supported. Its value is <b>NULL</b> .
pdml_status	character varying(8)	Specifies whether to enable a DML parallel execution in the current session.
port	numeric	Port number of the current session.

Name	Type	Description
process	character varying(24)	Process ID of the current session.
logon_time	date	Login time of the current session.
last_call_et	integer	Duration when the status of the current session changes last time.
osuser	character varying(128)	Not supported. Its value is <b>NULL</b> .
terminal	character varying(30)	Not supported. Its value is <b>NULL</b> .
type	character varying(10)	Not supported. Its value is <b>NULL</b> .
sql_address	raw	Not supported. Its value is <b>NULL</b> .
sql_hash_value	numeric	Not supported. Its value is <b>NULL</b> .
sql_child_number	numeric	Not supported. Its value is <b>NULL</b> .
sql_exec_id	numeric	Not supported. Its value is <b>NULL</b> .
prev_sql_address	raw	Not supported. Its value is <b>NULL</b> .
prev_hash_value	numeric	Not supported. Its value is <b>NULL</b> .
prev_sql_id	character varying(13)	Not supported. Its value is <b>NULL</b> .
prev_child_number	numeric	Not supported. Its value is <b>NULL</b> .
prev_exec_start	date	Not supported. Its value is <b>NULL</b> .
prev_exec_id	numeric	Not supported. Its value is <b>NULL</b> .
plsql_entry_object_id	numeric	Not supported. Its value is <b>NULL</b> .
plsql_entry_subprogram_id	numeric	Not supported. Its value is <b>NULL</b> .
plsql_object_id	numeric	Not supported. Its value is <b>NULL</b> .
plsql_subprogram_id	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
module	character varying(64)	Not supported. Its value is <b>NULL</b> .
module_hash	numeric	Not supported. Its value is <b>NULL</b> .
action	character varying(64)	Not supported. Its value is <b>NULL</b> .
action_hash	numeric	Not supported. Its value is <b>NULL</b> .
fixed_table_sequence	numeric	Not supported. Its value is <b>NULL</b> .
row_wait_object#	numeric	Not supported. Its value is <b>NULL</b> .
row_wait_file#	numeric	Not supported. Its value is <b>NULL</b> .
row_wait_block#	numeric	Not supported. Its value is <b>NULL</b> .
row_wait_row#	numeric	Not supported. Its value is <b>NULL</b> .
top_level_call#	numeric	Not supported. Its value is <b>NULL</b> .
pdml_enabled	character varying(3)	Not supported. Its value is <b>NULL</b> .
failover_type	character varying(13)	Not supported. Its value is <b>NULL</b> .
failover_method	character varying(10)	Not supported. Its value is <b>NULL</b> .
failed_over	character varying(3)	Not supported. Its value is <b>NULL</b> .
resource_consumer_group	character varying(32)	Not supported. Its value is <b>NULL</b> .
pddl_status	character varying(8)	Not supported. Its value is <b>NULL</b> .
pq_status	character varying(8)	Not supported. Its value is <b>NULL</b> .
current_queue_duration	numeric	Not supported. Its value is <b>NULL</b> .
client_identifier	character varying(64)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
blocking_session_status	character varying(11)	Not supported. Its value is <b>NULL</b> .
blocking_instance	numeric	Not supported. Its value is <b>NULL</b> .
blocking_session	numeric	Not supported. Its value is <b>NULL</b> .
final_blocking_session_status	character varying(11)	Not supported. Its value is <b>NULL</b> .
final_blocking_instance	numeric	Not supported. Its value is <b>NULL</b> .
final_blocking_session	numeric	Not supported. Its value is <b>NULL</b> .
seq#	numeric	Not supported. Its value is <b>NULL</b> .
event#	numeric	Not supported. Its value is <b>NULL</b> .
p1text	character varying(64)	Not supported. Its value is <b>NULL</b> .
p1	numeric	Not supported. Its value is <b>NULL</b> .
p1raw	raw	Not supported. Its value is <b>NULL</b> .
p2text	character varying(64)	Not supported. Its value is <b>NULL</b> .
p2	numeric	Not supported. Its value is <b>NULL</b> .
p2raw	raw	Not supported. Its value is <b>NULL</b> .
p3text	character varying(64)	Not supported. Its value is <b>NULL</b> .
p3	numeric	Not supported. Its value is <b>NULL</b> .
p3raw	raw	Not supported. Its value is <b>NULL</b> .
wait_class_id	numeric	Not supported. Its value is <b>NULL</b> .
wait_class#	numeric	Not supported. Its value is <b>NULL</b> .
wait_class	character varying(64)	Not supported. Its value is <b>NULL</b> .
wait_time	numeric	Not supported. Its value is <b>NULL</b> .
seconds_in_wait	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
state	character varying(19)	Not supported. Its value is <b>NULL</b> .
wait_time_micro	numeric	Not supported. Its value is <b>NULL</b> .
time_remaining_micro	numeric	Not supported. Its value is <b>NULL</b> .
time_since_last_wait_micro	numeric	Not supported. Its value is <b>NULL</b> .
service_name	character varying(64)	Not supported. Its value is <b>NULL</b> .
sql_trace	character varying(8)	Not supported. Its value is <b>NULL</b> .
sql_trace_waits	character varying(5)	Not supported. Its value is <b>NULL</b> .
sql_trace_binds	character varying(5)	Not supported. Its value is <b>NULL</b> .
sql_trace_plan_stats	character varying(10)	Not supported. Its value is <b>NULL</b> .
session_editon_id	numeric	Not supported. Its value is <b>NULL</b> .
creator_addr	raw	Not supported. Its value is <b>NULL</b> .
creator_serial#	numeric	Not supported. Its value is <b>NULL</b> .
ecid	character varying(64)	Not supported. Its value is <b>NULL</b> .
sql_translation_profile_id	numeric	Not supported. Its value is <b>NULL</b> .
pga_tunable_mem	numeric	Not supported. Its value is <b>NULL</b> .
shard_ddl_status	character varying(8)	Not supported. Its value is <b>NULL</b> .
con_id	numeric	Not supported. Its value is <b>NULL</b> .
external_name	character varying(1024)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
plsql_debugger_connected	character varying(5)	Not supported. Its value is <b>NULL</b> .

### 12.3.131 MPP\_TABLES

MPP\_TABLES displays information about tables in PGXC\_CLASS.

**Table 12-234** MPP\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains a table.
tablename	name	Table name.
tableowner	name	Table owner.
tablespace	name	Tablespace containing the table.
pgroup	name	Node group name.
nodeoids	oidvector_extend	List of distributed table node OIDs.

### 12.3.132 MY\_COL\_COMMENTS

MY\_COL\_COMMENTS displays column comments of the table accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-235** MY\_COL\_COMMENTS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.
comments	text	Comments.
origin_con_id	numeric	Not supported. Its value is <b>0</b> .

Name	Type	Description
schema	character varying(64)	Name of the namespace to which the column belongs.

### 12.3.133 MY\_COL\_PRIVS

MY\_COL\_PRIVS displays the column permission granting information of the current user as the object owner, grantor, or grantee. By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-236** MY\_COL\_PRIVS columns

Name	Type	Description
grantor	character varying(128)	Name of the user who grants the permission.
owner	character varying(128)	Object owner.
grantee	character varying(128)	Name of the user or role to which the permission is granted.
table_schema	character varying(128)	Schema of an object.
table_name	character varying(128)	Object name.
column_name	character varying(128)	Column name.
privilege	character varying(40)	Permission on a column.
grantable	character varying(3)	Specifies whether to grant privileges. <ul style="list-style-type: none"> <li>• YES</li> <li>• NO</li> </ul>
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.134 MY\_COLL\_TYPES

MY\_COLL\_TYPES displays information about types of collections created by the current user. By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-237** MY\_COLL\_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of a cluster.
type_name	character varying(128)	Name of a collection.
coll_type	character varying(128)	Description of a set.
upper_bound	numeric	Not supported. Its value is <b>NULL</b> .
elem_type_mod	character varying(7)	Not supported. Its value is <b>NULL</b> .
elem_type_name	character varying(128)	Name of the data type or user-defined type on which the collection is based.
length	numeric	Not supported. Its value is <b>NULL</b> .
precision	numeric	Not supported. Its value is <b>NULL</b> .
scale	numeric	Not supported. Its value is <b>NULL</b> .
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
elem_storage	character varying(7)	Not supported. Its value is <b>NULL</b> .
nulls_stored	character varying(3)	Not supported. Its value is <b>NULL</b> .
char_used	character varying(1)	Not supported. Its value is <b>NULL</b> .

### 12.3.135 MY\_CONS\_COLUMNS

MY\_CONS\_COLUMNS displays information about primary key constraint columns in tables accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-238** MY\_CONS\_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
table_name	character varying(64)	Name of a constraint-related table.

Name	Type	Description
column_name	character varying(64)	Name of a constraint-related column.
constraint_name	character varying(64)	Constraint name.
position	smallint	Position of a column in a table.

## 12.3.136 MY\_CONSTRAINTS

MY\_CONSTRAINTS displays table constraint information accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-239** MY\_CONSTRAINTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
constraint_name	vcharacter varying(64)	Constraint name.
constraint_type	text	Constraint type. <ul style="list-style-type: none"><li>● <b>c</b>: check constraint.</li><li>● <b>f</b>: foreign key constraint.</li><li>● <b>p</b>: primary key constraint.</li><li>● <b>u</b>: unique constraint.</li></ul>
table_name	character varying(64)	Name of a constraint-related table.
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint).
index_name	character varying(64)	Name of the constraint-related index (only for the unique constraint and primary key constraint).

## 12.3.137 MY\_DEPENDENCIES

MY\_DEPENDENCIES displays the dependencies between objects that are accessible to the current user. This view exists in both the PG\_CATALOG and SYS schema and all users can access this view.

**Table 12-240** MY\_DEPENDENCIES columns

Name	Type	Description
name	character varying(128)	Object name.
type	character varying(18)	Object type.
referenced_owner	character varying(128)	Owner of the referenced object.
referenced_name	character varying(128)	Name of the referenced object.
referenced_type	character varying(18)	Type of the referenced object.
referenced_link_name	character varying(128)	Name of the link to the parent object (if remote).
dependency_type	character varying(4)	Specifies whether the dependency is an REF dependency.

### 12.3.138 MY\_IND\_COLUMNS

MY\_IND\_COLUMNS displays column information about all indexes accessible to the current user. It is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-241** MY\_IND\_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	name	Column name.
column_position	smallint	Position of the column in the index.
column_length	numeric	Length of the column. For the variable-length type, its value is <b>NULL</b> .
char_length	numeric	Maximum length of a column, in bytes.

Name	Type	Description
descend	character varying	Specifies whether columns are sorted in descending ( <b>DESC</b> ) or ascending ( <b>ASC</b> ) order.
collated_column_id	numeric	Not supported. Its value is <b>NULL</b> .

### 12.3.139 MY\_IND\_EXPRESSIONS

MY\_IND\_EXPRESSIONS displays information about function-based expression indexes accessible to the current user. It is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-242** MY\_IND\_EXPRESSIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
column_expression	text	Function-based index expression of a specified column.
column_position	smallint	Position of the column in the index.

### 12.3.140 MY\_IND\_PARTITIONS

MY\_IND\_PARTITIONS displays the partition information about local indexes of level-1 partitioned table accessible to the current user (excluding global indexes of partitioned tables). It is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-243** MY\_IND\_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index to which an index partition belongs.

Name	Type	Description
index_name	character varying(64)	Name of the partitioned table index to which the index partition belongs.
partition_name	character varying(64)	Name of the index partition.
def_tablespace_name	name	Tablespace name of the index partition.
high_value	text	Upper limit of the partition corresponding to the index partition. <ul style="list-style-type: none"> <li>• For range partitioning and interval partitioning, the upper limit of each partition is displayed.</li> <li>• For list partitioning, the value list of each partition is displayed.</li> <li>• For hash partitioning, the number of each partition is displayed.</li> </ul>
index_partition_usable	Boolean	Specifies whether the index partition is available. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
schema	character varying(64)	Schema of the partitioned table index to which the index partition belongs.
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition.
composite	character varying(3)	Specifies whether the index is a local index on the level-2 partitioned table. This table does not store level-2 partition information. Therefore, the value is <b>NO</b> .

Name	Type	Description
subpartition_count	numeric	Number of level-2 partitions in a partition. This table does not store level-2 partition information. Therefore, the value is <b>0</b> .
partition_position	numeric	Position of an index partition in the index.
status	character varying(8)	Specifies whether the index partition is available.
tablespace_name	name	Name of the tablespace that contains the partition.
pct_free	numeric	Percentage of minimum available space in a block.
ini_trans	numeric	Initial number of transactions. The default value is <b>4</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is <b>128</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Specifies whether changes to an index are logged.
compression	character varying(13)	Specifies whether an index compression is enabled for a partitioned index.
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
clustering_factor	numeric	Sequence of a row in the table based on the value of the index. You need to run the <b>analyze</b> command to collect statistics.
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	timestamp with time zone	Last date when the partition was analyzed. Database restart is not supported. Otherwise, data loss will occur.
buffer_pool	character varying(7)	Actual buffer pool of a partition.
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
pct_direct_access	numeric	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table. The distributed system does not support interval partitioning. Therefore, the value of this parameter is <b>NO</b> .
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
orphaned_entries	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.141 MY\_IND\_SUBPARTITIONS

MY\_IND\_SUBPARTITIONS displays the partition information about local indexes of level-2 partitioned table owned by the current user (excluding global indexes of partitioned tables). It is accessible to all users. This view exists in both PG\_CATALOG and SYS schema. The distributed system does not support level-2 partitions. Currently, all columns in this table are set to **NULL**.

Table 12-244 MY\_IND\_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
index_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
high_value	text	Not supported. Its value is <b>NULL</b> .
high_value_length	numeric	Not supported. Its value is <b>NULL</b> .
partition_position	numeric	Not supported. Its value is <b>NULL</b> .
subpartition_position	numeric	Not supported. Its value is <b>NULL</b> .
status	character varying(8)	Not supported. Its value is <b>NULL</b> .
tablespace_name	character varying(30)	Not supported. Its value is <b>NULL</b> .
pct_free	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Not supported. Its value is <b>NULL</b> .
max_trans	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
compression	character varying(13)	Not supported. Its value is <b>NULL</b> .
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
clustering_factor	numeric	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .
buffer_pool	character varying(7)	Not supported. Its value is <b>NULL</b> .
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Not supported. Its value is <b>NULL</b> .
segment_created	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .

## 12.3.142 MY\_INDEXES

MY\_INDEXES displays index information about the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-245** MY\_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_name	character varying(64)	Name of the table corresponding to the index.
uniqueness	text	Specifies whether the index is a unique index. <ul style="list-style-type: none"><li>• <b>UNIQUE</b>: unique index.</li><li>• <b>NONUNIQUE</b>: non-unique index.</li></ul>
partitioned	character(3)	Specifies whether the index has the property of partitioned tables. <ul style="list-style-type: none"><li>• <b>Yes</b>: The index has the property of a partitioned table.</li><li>• <b>No</b>: The index does not have the property of a partitioned table.</li></ul>
generated	character varying(1)	Specifies whether the index name is generated by the system. <ul style="list-style-type: none"><li>• <b>y</b>: The index name is generated by the system.</li><li>• <b>n</b>: The index name is not generated by the system.</li></ul>

Name	Type	Description
index_type	character varying(27)	Index type. <ul style="list-style-type: none"> <li>• <b>NORMAL</b>: Index attributes are simple references, and the expression tree is empty.</li> <li>• <b>FUNCTION-BASED NORMAL</b>: Expression trees are used for index attributes that are not simple column references.</li> </ul>
table_owner	character varying(128)	Owner of an index object.
table_type	character(11)	Type of the index object. <ul style="list-style-type: none"> <li>• <b>TABLE</b>: The index object is of the table type.</li> </ul>
tablespace_name	character varying(30)	Name of a tablespace that contains the index.
status	character varying(8)	Status of a non-partitioned index. <ul style="list-style-type: none"> <li>• <b>VALID</b>: Non-partitioned indexes can be used for query.</li> <li>• <b>UNUSABLE</b>: The non-partitioned index is unavailable.</li> <li>• <b>N/A</b>: The index has the property of a partitioned table.</li> </ul>
compression	character varying(13)	Not supported. Its value is <b>NULL</b> .
prefix_length	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Not supported. Its value is <b>NULL</b> .
max_trans	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
pct_threshold	numeric	Not supported. Its value is <b>NULL</b> .
include_column	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
pct_free	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(3)	Not supported. Its value is <b>NULL</b> .
blevel	numeric	Not supported. Its value is <b>NULL</b> .
leaf_blocks	numeric	Not supported. Its value is <b>NULL</b> .
distinct_keys	numeric	Not supported. Its value is <b>NULL</b> .
avg_leaf_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
avg_data_blocks_per_key	numeric	Not supported. Its value is <b>NULL</b> .
clustering_factor	numeric	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
degree	character varying(40)	Not supported. Its value is <b>NULL</b> .
instances	character varying(40)	Not supported. Its value is <b>NULL</b> .
temporary	character varying(1)	Not supported. Its value is <b>NULL</b> .
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
buffer_pool	character varying(7)	Not supported. Its value is <b>NULL</b> .
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
duration	character varying(15)	Not supported. Its value is <b>NULL</b> .
pct_direct_access	numeric	Not supported. Its value is <b>NULL</b> .
ityp_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
ityp_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
parameters	character varying(1000)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
domidx_status	character varying(12)	Not supported. Its value is <b>NULL</b> .
domidx_opstatus	character varying(6)	Not supported. Its value is <b>NULL</b> .
funcidx_status	character varying(8)	Not supported. Its value is <b>NULL</b> .
join_index	character varying(3)	Not supported. Its value is <b>NULL</b> .
iot_redundant_pkey_elim	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
dropped	character varying(3)	Not supported. Its value is <b>NULL</b> .
visibility	character varying(9)	Not supported. Its value is <b>NULL</b> .
domidx_management	character varying(14)	Not supported. Its value is <b>NULL</b> .
segment_created	character varying(3)	Not supported. Its value is <b>NULL</b> .
orphaned_entries	character varying(3)	Not supported. Its value is <b>NULL</b> .
indexing	character varying(7)	Not supported. Its value is <b>NULL</b> .
auto	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.143 MY\_JOBS

MY\_JOBS displays details about the scheduled jobs owned by the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-246** MY\_JOBS columns

Name	Type	Description
job	bigint	Job ID.
log_user	name	Username of a job creator.
priv_user	name	Username of a job executor.
dbname	name	Name of the database where the job is created.
schema_user	name	Default schema name of a scheduled job.
start_date	timestamp without time zone	Time when a job starts to be executed for the first time.
start_suc	text	Start time of the first successful job execution.
last_date	timestamp without time zone	Start time of the last job execution.
last_suc	text	Start time of the last successful job execution.

Name	Type	Description
last_sec	text	Start time of the last successful job execution. Compatibility is supported.
this_date	timestamp without time zone	Start time of an ongoing job execution.
this_suc	text	Start time of an ongoing job execution.
this_sec	text	Start time of an ongoing job execution. Compatibility is supported.
next_date	timestamp without time zone	Schedule time of the next job execution.
next_suc	text	Schedule time of the next job execution.
next_sec	text	Schedule time of the next job execution. Compatibility is supported.
total_time	numeric	Latest execution duration of a job.
broken	text	If the value of the <b>status</b> column is 'd', the value is 'y'. Otherwise, the value is 'n'.
status	"char"	Status of the current job. The value can be 'r', 's', 'f', or 'd'. The default value is 'r'. <ul style="list-style-type: none"><li>● r: running</li><li>● s: successful execution</li><li>● f: failed execution</li><li>● d: canceling an execution</li></ul>
interval	text	Time expression used to calculate the next time the job will be executed. If this parameter is set to <b>null</b> , the job will be executed once only.
failures	smallint	Number of times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
what	text	Executable job.

Name	Type	Description
nls_env	character varying(4000)	Not supported. Its value is <b>NULL</b> .
misc_env	raw	Not supported. Its value is <b>NULL</b> .
instance	numeric	Not supported. Its value is <b>NULL</b> .

## 12.3.144 MY\_OBJECTS

MY\_OBJECTS displays database objects accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-247** MY\_OBJECTS columns

Name	Type	Description
object_name	name	Object name.
object_id	oid	Object OID.
object_type	name	Type of the object ( <b>TABLE</b> , <b>INDEX</b> , <b>SEQUENCE</b> , <b>VIEW</b> , or <b>GLOBAL SECONDARY INDEX</b> ).
namespace	oid	Namespace that the object belongs to.
temporary	character(1)	Specifies whether the object is a temporary object.
status	character varying(7)	Object status.
subobject_name	name	Subobject name of an object.
generated	character(1)	Specifies whether the object name is generated by the system.
created	timestamp with time zone	Time when the object was created.
last_ddl_time	timestamp with time zone	Last time when the object was modified.
default_collation	character varying(100)	Default collation of objects.
data_object_id	numeric	Not supported. Its value is <b>NULL</b> .
timestamp	character varying(19)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
edition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
sharing	character varying(18)	Not supported. Its value is <b>NULL</b> .
editionable	character varying(1)	Not supported. Its value is <b>NULL</b> .
oracle_maintained	character varying(1)	Not supported. Its value is <b>NULL</b> .
application	character varying(1)	Not supported. Its value is <b>NULL</b> .
duplicated	character varying(1)	Not supported. Its value is <b>NULL</b> .
sharded	character varying(1)	Not supported. Its value is <b>NULL</b> .
created_appid	numeric	Not supported. Its value is <b>NULL</b> .
modified_appid	numeric	Not supported. Its value is <b>NULL</b> .
created_vsnid	numeric	Not supported. Its value is <b>NULL</b> .
modified_vsnid	numeric	Not supported. Its value is <b>NULL</b> .

**NOTICE**

For details about the value ranges of **created** and **last\_ddl\_time**, see [PG\\_OBJECT](#).

### 12.3.145 MY\_PART\_COL\_STATISTICS

MY\_PART\_COL\_STATISTICS displays column statistics and histogram information about table partitions owned by the current user. It is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-248** MY\_PART\_COL\_STATISTICS columns

Name	Type	Description
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Table partition name.

Name	Type	Description
column_name	character varying(4000)	Column name.
num_distinct	numeric	Not supported. Its value is <b>NULL</b> .
low_value	raw	Not supported. Its value is <b>NULL</b> .
high_value	raw	Not supported. Its value is <b>NULL</b> .
density	numeric	Not supported. Its value is <b>NULL</b> .
num_nulls	numeric	Not supported. Its value is <b>NULL</b> .
num_buckets	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	date	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
notes	character varying(63)	Not supported. Its value is <b>NULL</b> .
avg_col_len	numeric	Not supported. Its value is <b>NULL</b> .
histogram	character varying(15)	Not supported. Its value is <b>NULL</b> .
schema	character varying(64)	Name of the namespace to which the column belongs.

### 12.3.146 MY\_PART\_INDEXES

**MY\_PART\_INDEXES** displays information about partitioned table indexes accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-249** MY\_PART\_INDEXES columns

Name	Type	Description
def_tablespace_name	name	Tablespace name of the partitioned table index
index_owner	character varying(64)	Name of the owner of a partitioned table index
index_name	character varying(64)	Name of the partitioned table index
partition_count	bigint	Number of index partitions of the partitioned table index
partitioning_key_count	integer	Number of partition keys of the partitioned table
partitioning_type	text	Partition policy of the partitioned table <b>NOTE</b> For details about the supported partitioning policies of the current partitioned table, see <a href="#">CREATE TABLE PARTITION</a> .
schema	character varying(64)	Schema of the partitioned table index
table_name	character varying(64)	Name of the partitioned table to which the partitioned table index belongs

### 12.3.147 MY\_PART\_KEY\_COLUMNS

MY\_PART\_KEY\_COLUMNS displays information about the partition key columns of partitioned tables or partitioned indexes owned by the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-250** MY\_PART\_KEY\_COLUMNS columns

Name	Type	Description
name	character varying(128)	Name of a partitioned table or index.
object_type	character varying(128)	Valid values are <b>table</b> and <b>index</b> . If the partition is a partitioned table, the value is <b>table</b> . If the partition is a partitioned index, the value is <b>index</b> .

Name	Type	Description
column_name	character varying(4000)	Partition key column name of a partitioned table or index.
column_position	numeric	Position of a column in a partition.
collated_column_id	numeric	Not supported. Its value is <b>NULL</b> .

## 12.3.148 MY\_PART\_TABLES

MY\_PART\_TABLES displays information about partitioned tables accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-251** MY\_PART\_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table.
table_name	character varying(64)	Name of the partitioned table.
partitioning_type	text	Partitioning policy of the partitioned table. <b>NOTE</b> For details about the supported partitioning policies of the current partitioned table, see <a href="#">CREATE TABLE PARTITION</a> .
partition_count	bigint	Number of partitions of the partitioned table.
partitioning_key_count	integer	Number of partition keys of the partitioned table.
def_tablespace_name	name	Tablespace name of the partitioned table.
schema	character varying(64)	Schema of the partitioned table.
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, <b>NONE</b> is displayed. The distributed system does not support level-2 partitions. Set this parameter to <b>NONE</b> .

Name	Type	Description
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is <b>1</b> for a level-2 partitioned table and <b>0</b> for a level-1 partitioned table. The distributed system does not support level-2 partitions. Set this parameter to <b>0</b> .
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table. The distributed system does not support level-2 partitions. Set this parameter to <b>0</b> .
status	character varying(8)	Not supported. Its value is <b>valid</b> .
def_pct_free	numeric	Default value of <b>PCTFREE</b> used when a partition is added.
def_pct_used	numeric	Not supported. Its value is <b>NULL</b> .
def_ini_trans	numeric	Default value of <b>INITRANS</b> used when a partition is added.
def_max_trans	numeric	Default value of <b>MAXTRANS</b> used when a partition is added.
def_initial_extent	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_next_extent	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_min_extents	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_max_extents	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_max_size	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_pct_increase	character varying(40)	Not supported. Its value is <b>NULL</b> .
def_freelists	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
def_freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
def_logging	character varying(7)	Not supported. Its value is <b>NULL</b> .
def_compression	character varying(8)	Default compression mode used when a partition is added. Value range: <ul style="list-style-type: none"> <li>• NONE</li> <li>• ENABLED</li> <li>• DISABLED</li> </ul>
def_compress_for	character varying(30)	Default compression mode used when a partition is added. <b>NOTE</b> For available compression modes and compression levels, see <a href="#">WITH ( { storage_paramet...</a>
def_buffer_pool	character varying(7)	Not supported. Its value is <b>DEFAULT</b> .
def_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
def_cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
ref_ptn_constraint_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
interval	character varying(1000)	Interval.
autolist	character varying(3)	Not supported. Its value is <b>NO</b> .
interval_subpartition	character varying(1000)	Not supported. Its value is <b>NULL</b> .
autolist_subpartition	character varying(3)	Not supported. Its value is <b>NO</b> .
is_nested	character varying(3)	Not supported. Its value is <b>NO</b> .
def_segment_creation	character varying(4)	Currently, the segment-page mode is not supported. When the segment-page mode is enabled, the value is <b>YES</b> .
def_indexing	character varying(3)	Not supported. Its value is <b>ON</b> .
def_inmemory	character varying(8)	Not supported. Its value is <b>NONE</b> .

Name	Type	Description
def_inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
def_inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
def_inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
def_inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
def_read_only	character varying(3)	Not supported. Its value is <b>NO</b> .
def_cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .

## 12.3.149 MY\_PROCEDURES

MY\_PROCEDURES displays information about stored procedures, functions, or triggers owned by the current user. This view exists in both PG\_CATALOG and SYS schema. This view can be accessed by all users. Only the information about the current user can be viewed.

**Table 12-252** MY\_PROCEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure, function, or trigger.
object_name	character varying(64)	Name of a stored procedure, function, or trigger.
procedure_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
object_id	oid	OID of a stored procedure, function, or trigger.
subprogram_id	numeric	Not supported. Its value is <b>NULL</b> .
overload	character varying(40)	$n$ th overloaded function of the name.

Name	Type	Description
object_type	character varying(13)	Object type name.
aggregate	character varying(3)	Specifies whether the function is an aggregate function: <ul style="list-style-type: none"><li>• <b>YES</b></li><li>• <b>NO</b></li></ul>
pipelined	character varying(3)	Not supported. Its value is <b>NO</b> .
impltypeowner	character varying(128)	Owner of an implementation type.
impltypename	character varying(128)	Name of an implementation type.
parallel	character varying(3)	Not supported. Its value is <b>NO</b> .
interface	character varying(3)	Not supported. Its value is <b>NO</b> .
deterministic	character varying(3)	Not supported. Its value is <b>NO</b> .
authid	character varying(12)	Specifies whether to use the creator permission or caller permission: <ul style="list-style-type: none"><li>• <b>DEFINER</b>: The creator permission is used.</li><li>• <b>CURRENT_USER</b>: The caller permission is used.</li></ul> This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
result_cache	character varying(3)	Not supported. Its value is <b>NULL</b> .
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .
polymorphic	character varying(5)	Not supported. Its value is <b>NULL</b> .
argument_number	smallint	Number of input parameters in a stored procedure.

### 12.3.150 MY\_ROLE\_PRIVS

MY\_ROLE\_PRIVS displays permission information about roles (including the public role) granted to the current user. By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-253 MY\_ROLE\_PRIVS columns**

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
granted_role	character varying(128)	Role to be granted.
admin_option	character varying(3)	Specifies whether the grant contains the <b>ADMIN</b> option. <ul style="list-style-type: none"> <li>• <b>YES</b></li> <li>• <b>NO</b></li> </ul>
delegate_option	character varying(3)	Not supported. Its value is <b>NULL</b> .
default_role	character varying(3)	Not supported. Its value is <b>NULL</b> .
os_granted	character varying(3)	Not supported. Its value is <b>NULL</b> .
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.151 MY\_SCHEDULER\_JOB\_ARGS

MY\_SCHEDULER\_JOB\_ARG displays the parameters related to the jobs owned by the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-254 MY\_SCHEDULER\_JOB\_ARGS columns**

Name	Type	Description
job_name	character varying(128)	Name of the job to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of the parameter.

Name	Type	Description
value	character varying(4000)	Parameter value.
anydata_value	character varying(4000)	Not supported. Its value is <b>NULL</b> .
out_argument	character varying(5)	Reserved column. Its value is <b>NULL</b> .

### 12.3.152 MY\_SCHEDULER\_PROGRAM\_ARGS

MY\_SCHEDULER\_PROGRAM\_ARG displays the parameters related to the programs owned by the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-255** MY\_SCHEDULER\_PROGRAM\_ARGS columns

Name	Type	Description
program_name	character varying(128)	Name of the program to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of the parameter.
metadata_attribute	character varying(19)	Not supported. Its value is <b>NULL</b> .
default_value	character varying(4000)	Default parameter value.
default_anydata_value	character varying(4000)	Not supported. Its value is <b>NULL</b> .
out_argument	character varying(5)	Reserved column. Its value is <b>NULL</b> .

### 12.3.153 MY\_SEQUENCES

MY\_SEQUENCES displays the sequence information about the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-256** MY\_SEQUENCES columns

Name	Type	Description
sequence_owner	character varying(64)	Owner of a sequence.
sequence_name	character varying(64)	Name of the sequence.
min_value	int16	Minimum value of the sequence.
max_value	int16	Maximum value of the sequence.
increment_by	int16	Value by which the sequence is incremented.
cycle_flag	character(1)	Specifies whether the sequence is a cyclic sequence. The value can be <b>Y</b> or <b>N</b> . <ul style="list-style-type: none"><li>• <b>Y</b>: It is a cyclic sequence.</li><li>• <b>N</b>: It is not a cyclic sequence.</li></ul>
last_number	int16	Value of the previous sequence.
cache_size	int16	Size of the sequence disk cache.
order_flag	character varying(1)	Specifies whether a sequence occurs in a request sequence. This parameter is not supported. Set it to <b>NULL</b> .
scale_flag	character varying(1)	Specifies whether a sequence is a scalable sequence. This parameter is not supported. Set it to <b>NULL</b> .
extend_flag	character varying(1)	Specifies whether the value generated by a scalable sequence exceeds the maximum or minimum value of the sequence. Not supported. Its value is <b>NULL</b> .
sharded_flag	character varying(1)	Specifies whether a sequence is a shard sequence. Not supported. Its value is <b>NULL</b> .
session_flag	character varying(1)	Specifies whether a sequence is a private session. This parameter is not supported. Its value is <b>NULL</b> .

Name	Type	Description
keep_value	character varying(1)	Specifies whether to retain the sequence value during replay after a failure. Not supported. Its value is <b>NULL</b> .

### 12.3.154 MY\_SOURCE

MY\_SOURCE displays the definition information about stored procedures, functions, and triggers accessible to the current user. This view exists in both PG\_CATALOG and SYS schema.

Table 12-257 MY\_SOURCE columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	name	Object type. The value can be <b>function</b> , <b>procedure</b> , or <b>trigger</b> .
line	numeric	Number of the source line.
text	text	Text source of the storage object.
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .

### 12.3.155 MY\_SUBPART\_KEY\_COLUMNS

MY\_SUBPART\_KEY\_COLUMNS displays information about the partition key columns of level-2 partitioned tables or partitioned indexes owned by the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in both PG\_CATALOG and SYS schema. The distributed system does not support level-2 partitioned tables. All columns in this view are set to **NULL**.

Table 12-258 MY\_SUBPART\_KEY\_COLUMNS columns

Name	Type	Description
name	character varying(128)	Not supported. Its value is <b>NULL</b> .
object_type	character varying(128)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
column_name	character varying(4000)	Not supported. Its value is <b>NULL</b> .
column_position	numeric	Not supported. Its value is <b>NULL</b> .
collated_column_id	numeric	Not supported. Its value is <b>NULL</b> .

## 12.3.156 MY\_SYNONYMS

MY\_SYNONYMS displays synonyms in the current schema. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-259** MY\_SYNONYMS columns

Name	Type	Description
schema_name	text	Name of the schema to which the synonym belongs.
synonym_name	text	Synonym name.
table_owner	text	Owner of the associated object. Although the column is called <b>table_owner</b> , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_name	text	Name of the associated object. Although the column is called <b>table_name</b> , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_schema_name	text	Schema name of the associated object. Although the column is called <b>table_schema_name</b> , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
db_link	character varying(128)	Reserved column. Its value is <b>NULL</b> .

Name	Type	Description
origin_con_id	character varying(256)	Not supported. Its value is <b>0</b> .

### 12.3.157 MY\_SYS\_PRIVS

MY\_SYS\_PRIVS displays information about system permissions granted to the current user. By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-260** MY\_SYS\_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
privilege	character varying(40)	System permission
admin_option	character varying(3)	Specifies whether the grant contains the <b>ADMIN</b> option. <ul style="list-style-type: none"><li>• <b>YES</b></li><li>• <b>NO</b></li></ul>
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.158 MY\_TAB\_COL\_STATISTICS

MY\_TAB\_COL\_STATISTICS displays column statistics and histogram information extracted from MY\_TAB\_COLUMNS. All users can access this view. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-261** MY\_TAB\_COL\_STATISTICS columns

Name	Type	Description
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Low value in a column.

Name	Type	Description
high_value	raw	High value in a column.
density	numeric	<ul style="list-style-type: none"> <li>If there is a histogram on <b>COLUMN_NAME</b>, this column displays the selectivity of values in the histogram that span less than two endpoints. It does not represent the selectivity of values that span two or more endpoints.</li> <li>If no histogram is available on <b>COLUMN_NAME</b>, the value of this column is <b>1/NUM_DISTINCT</b>.</li> </ul>
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
sample_size	numeric	Sample size used to analyze a column.
last_analyzed	timestamp(0) without time zone	Date when a column was last analyzed. Database restart is not supported. Otherwise, data loss will occur.
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
notes	character varying(99)	Not supported. Its value is <b>NULL</b> .
avg_col_len	numeric	Average length of a column, in bytes.
histogram	character varying(15)	<p>Specifies whether the histogram exists and the type of the histogram if it exists.</p> <ul style="list-style-type: none"> <li><b>NONE</b>: no histogram.</li> <li><b>FREQUENCY</b>: frequency histogram.</li> <li><b>EQUI-WIDTH</b>: equal-width histogram.</li> </ul>
scope	character varying(7)	For statistics collected on any table other than global temporary tables, the value is <b>SHARED</b> (indicating that the statistics are shared among all sessions).
schema	character varying(64)	Name of the namespace to which the column belongs.

## 12.3.159 MY\_TAB\_COLUMNS

MY\_TAB\_COLUMNS displays the columns of the tables and views owned by the current user. This view exists in both PG\_CATALOG and SYS schema. All users can access this view. Only the information about the user is displayed.

**Table 12-262** MY\_TAB\_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	character varying(64)	Column name.
data_type	character varying(128)	Data type of the column.
data_type_mod	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_type_owner	character varying(128)	Owner of the data type of a column.
data_length	integer	Length of the column, in bytes.
data_precision	integer	Precision of a data type. This column is valid for the numeric data type and <b>NULL</b> for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and <b>0</b> for other data types.
nullable	bpchar	Specifies whether a column can be empty ( <b>n</b> for a primary key constraint or NOT NULL constraint).
column_id	integer	Sequence number of the column when the table is created.
default_length	numeric	Length of the default value of a column, in bytes.
data_default	text	Default value of a column.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Minimum value in a column.
high_value	raw	Maximum value in a column.

Name	Type	Description
density	numeric	Column density.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	date	Last analysis date.
sample_size	numeric	Sample size used to analyze a column.
character_set_name	character varying(44)	Not supported. Its value is <b>NULL</b> .
char_col_decl_length	numeric	Declaration length of a column of the character type.
global_stats	character varying(3)	Not supported. Its value is <b>NO</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NO</b> .
avg_col_len	numeric	Average length of a column, in bytes.
char_length	numeric	Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types.
char_used	character varying(1)	Not supported. Set it to <b>B</b> if the data type is varchar, nvarchar2, bpchar, or char, and to <b>NULL</b> for other data types.
v80_fmt_image	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_upgraded	character varying(3)	Not supported. Its value is <b>YES</b> .
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram if it exists. <ul style="list-style-type: none"><li>• <b>NONE</b>: no histogram.</li><li>• <b>FREQUENCY</b>: frequency histogram.</li><li>• <b>EQUI_WIDTH</b>: equal-width histogram.</li></ul>

Name	Type	Description
default_on_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
identity_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
evaluation_edition	character varying(128)	Not supported. Its value is <b>NULL</b> .
unusable_before	character varying(128)	Not supported. Its value is <b>NULL</b> .
unusable_beginning	character varying(128)	Not supported. Its value is <b>NULL</b> .
collation	character varying(100)	Collation rule of a column. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
comments	text	Comment of a column.
schema	character varying(64)	Name of the namespace to which the column belongs.

### 12.3.160 MY\_TAB\_COMMENTS

MY\_TAB\_COMMENTS displays comments on all tables and views owned by the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-263** MY\_TAB\_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view.
table_name	character varying(64)	Name of the table or view.
comments	text	Comments.
schema	character varying(64)	Name of the namespace to which the table belongs.

## 12.3.161 MY\_TAB\_HISTOGRAMS

MY\_TAB\_HISTOGRAMS displays histogram information about the tables or views owned by the current user. All users can access this view. This view exists in both PG\_CATALOG and SYS schema.

Name	Type	Description
table_name	character varying(128)	Table name.
column_name	character varying(4000)	Column name.
endpoint_number	numeric	Bucket ID of the histogram.
endpoint_value	numeric	Not supported. Its value is <b>NULL</b> .
endpoint_actual_value	character varying(4000)	Actual value of the bucket endpoint.
endpoint_actual_value_raw	raw	Not supported. Its value is <b>NULL</b> .
endpoint_repeat_count	numeric	Not supported. Its value is <b>NULL</b> .
scope	character varying(7)	Not supported. Its value is <b>SHARED</b> .

## 12.3.162 MY\_TAB\_PARTITIONS

MY\_TAB\_PARTITIONS displays information about level-1 table partitions accessible to the current user (including level-2 partitioned tables). Each level-1 table partition of a partitioned table accessible to the current user has one record in MY\_TAB\_PARTITIONS. It is accessible to all users. This view exists in both PG\_CATALOG and SYS schema. The distributed system does not support level-2 partitions. Therefore, this view does not store level-1 partition information of level-2 partitioned tables.

**Table 12-264** MY\_TAB\_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.

Name	Type	Description
high_value	text	Limit of a partition. <ul style="list-style-type: none"> <li>For range partitioning and interval partitioning, the upper limit of each partition is displayed.</li> <li>For list partitioning, the value list of each partition is displayed.</li> <li>For hash partitioning, the number of each partition is displayed.</li> </ul>
tablespace_name	name	Tablespace name of the partitioned table.
schema	character varying(64)	Name of a namespace.
composite	character varying(3)	Specifies whether the table is a level-2 partitioned table.
subpartition_count	numeric	Not supported. Its value is <b>NULL</b> .
high_value_length	integer	Length of the binding value expression in a partition.
partition_position	numeric	Position of the partition in the table.
pct_free	numeric	Minimum percentage of available space in a block.
pct_used	numeric	Not supported. Its value is <b>NULL</b> .
ini_trans	numeric	Initial number of transactions. The default value is <b>4</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is <b>128</b> . The value is <b>NULL</b> for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_extent	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .
logging	character varying(7)	Specifies whether changes to the table are logged.
compression	character varying(8)	Actual compression attribute of a partitioned table.
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
num_rows	numeric	Not supported. Its value is <b>NULL</b> .
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_row_len	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Not supported. Its value is <b>NULL</b> .
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool allocated to a partitioned block.
flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
cell_flash_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
is_nested	character varying(3)	Specifies whether this partitioned table is a nested partitioned table.
parent_table_partition	character varying(128)	Not supported. Its value is <b>NULL</b> .
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table. The distributed system does not support interval partitioning. Therefore, the value of this parameter is <b>NO</b> .

Name	Type	Description
segment_created	character varying(4)	Specifies whether a partitioned table has segments created or has not been created.
indexing	character varying(4)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(4)	Not supported. Its value is <b>NULL</b> .
inmemory	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
inmemory_service_name	character varying(100)	Not supported. Its value is <b>NULL</b> .
memoptimize_read	character varying(8)	Not supported. Its value is <b>NULL</b> .
memoptimize_write	character varying(8)	Not supported. Its value is <b>NULL</b> .

### 12.3.163 MY\_TAB\_STATISTICS

MY\_TAB\_STATISTICS displays statistics about tables owned by the current user in the database. This view exists in both PG\_CATALOG and SYS schema and all users can access this view.

**Table 12-265** MY\_TAB\_STATISTICS columns

Name	Type	Description
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
partition_position	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
subpartition_position	numeric	Not supported. Its value is <b>NULL</b> .
object_type	character varying(12)	Object type. <ul style="list-style-type: none"><li>• TABLE</li><li>• PARTITION</li><li>• SUBPARTITION</li></ul>
num_rows	numeric	Number of rows in an object.
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_row_len	numeric	Average row length, including the row overhead.
avg_space_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .
num_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_cached_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_cache_hit_ratio	numeric	Not supported. Its value is <b>NULL</b> .
im_imcu_count	numeric	Not supported. Its value is <b>NULL</b> .
im_block_count	numeric	Not supported. Its value is <b>NULL</b> .
im_stat_update_time	timestamp(9) without time zone	Not supported. Its value is <b>NULL</b> .
scan_rate	numeric	Not supported. Its value is <b>NULL</b> .
sample_size	numeric	Number of samples used for analyzing a table.
last_analyzed	timestamp with time zone	Date when a table was last analyzed. Database restart is not supported. Otherwise, data loss will occur.
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
stattype_locked	character varying(5)	Not supported. Its value is <b>NULL</b> .
stale_stats	character varying(7)	Not supported. Its value is <b>NULL</b> .
notes	character varying(25)	Not supported. Its value is <b>NULL</b> .
scope	character varying(7)	Not supported. The default value is <b>SHARED</b> .

### 12.3.164 MY\_TAB\_STATS\_HISTORY

MY\_TAB\_STATS\_HISTORY provides the table statistics history of the tables owned by the current user. All users can access this view. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-266** MY\_TAB\_STATS\_HISTORY columns

Name	Type	Description
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
subpartition_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
stats_update_time	timestamp(6) with time zone	Time when statistics are updated. Database restart is not supported. Otherwise, data loss will occur.

### 12.3.165 MY\_TABLES

MY\_TABLES displays information about the tables owned by the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-267** MY\_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
tablespace_name	character varying(64)	Tablespace name of the table.

Name	Type	Description
dropped	character varying	Specifies whether the current record is deleted. <ul style="list-style-type: none"> <li>• <b>yes</b>: deleted.</li> <li>• <b>no</b>: not deleted.</li> </ul>
num_rows	numeric	Estimated number of rows in the table.
status	character varying(8)	Specifies whether the current record is valid. <ul style="list-style-type: none"> <li>• <b>valid</b>: valid.</li> </ul>
sample_size	numeric	Number of samples used for analyzing the table.
temporary	character(1)	Specifies whether the table is a temporary table. <ul style="list-style-type: none"> <li>• <b>y</b>: The table is a temporary table.</li> <li>• <b>n</b>: The table is not a temporary table.</li> </ul>
pct_free	numeric	Minimum percentage of free space in a block.
ini_trans	numeric	Initial number of transactions.
max_trans	numeric	Maximum number of transactions.
avg_row_len	integer	Average number of bytes in each row.
partitioned	character varying(3)	Specifies whether a table is a partitioned table.
last_analyzed	timestamp with time zone	Last time when the table was analyzed. Database restart is not supported. Otherwise, data loss will occur.
row_movement	character varying(8)	Specifies whether to allow partition row movement.
compression	character varying(8)	Specifies whether to enable a table compression.
duration	character varying(15)	Time elapsed when a temporary table is processed.
logical_replication	character varying(8)	Specifies whether logical replication is enabled for a table.

Name	Type	Description
external	character varying(3)	Specifies whether the table is a foreign table.
logging	character varying(3)	Specifies whether changes to a table are logged.
default_collation	character varying(100)	Default collation of a table.
degree	character varying(10)	Number of instances in a scanned table.
table_lock	character varying(8)	Specifies whether to enable a table lock.
nested	character varying(3)	Specifies whether a table is a nested table.
buffer_pool	character varying(7)	Default buffer pool of a table.
flash_cache	character varying(7)	Smart flash cache hint in database for a table block.
cell_flash_cache	character varying(7)	Cell flash cache hint for a table block.
skip_corrupt	character varying(8)	Specifies whether to skip corrupted blocks during table scanning.
has_identity	character varying(3)	Specifies whether a table has an identifier column.
segment_created	character varying(3)	Specifies whether a table segment has been created.
monitoring	character varying(3)	Specifies whether to monitor the modification of a table.
cluster_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
iot_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
pct_used	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
freelists	numeric	Not supported. Its value is <b>NULL</b> .
freelist_groups	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
backed_up	character varying(1)	Not supported. Its value is <b>NULL</b> .
blocks	numeric	Not supported. Its value is <b>NULL</b> .
empty_blocks	numeric	Not supported. Its value is <b>NULL</b> .
avg_space	numeric	Not supported. Its value is <b>NULL</b> .
chain_cnt	numeric	Not supported. Its value is <b>NULL</b> .
avg_space_freelists_blocks	numeric	Not supported. Its value is <b>NULL</b> .
num_freelist_blocks	numeric	Not supported. Its value is <b>NULL</b> .
instances	character varying(10)	Not supported. Its value is <b>NULL</b> .
cache	character varying(5)	Not supported. Its value is <b>NULL</b> .
iot_type	character varying(12)	Not supported. Its value is <b>NULL</b> .
secondary	character varying(1)	Not supported. Its value is <b>NULL</b> .
global_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
user_stats	character varying(3)	Not supported. Its value is <b>NULL</b> .
cluster_owner	character varying(30)	Not supported. Its value is <b>NULL</b> .
dependencies	character varying(8)	Not supported. Its value is <b>NULL</b> .
compression_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(3)	Not supported. Its value is <b>NULL</b> .
result_cache	character varying(7)	Not supported. Its value is <b>NULL</b> .
clustering	character varying(3)	Not supported. Its value is <b>NULL</b> .
activity_tracking	character varying(23)	Not supported. Its value is <b>NULL</b> .
dml_timestamp	character varying(25)	Not supported. Its value is <b>NULL</b> .
container_data	character varying(3)	Not supported. Its value is <b>NULL</b> .
inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
duplicated	character varying(1)	Not supported. Its value is <b>NULL</b> .
sharded	character varying(1)	Not supported. Its value is <b>NULL</b> .
hybrid	character varying(3)	Not supported. Its value is <b>NULL</b> .
cellmemory	character varying(24)	Not supported. Its value is <b>NULL</b> .
containers_default	character varying(3)	Not supported. Its value is <b>NULL</b> .
container_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .
container_map_object	character varying(3)	Not supported. Its value is <b>NULL</b> .
memoptimize_read	character varying(8)	Not supported. Its value is <b>NULL</b> .
memoptimize_write	character varying(8)	Not supported. Its value is <b>NULL</b> .
has_sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
admit_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
data_link_dml_enabled	character varying(3)	Not supported. Its value is <b>NULL</b> .
object_id_type	character varying(16)	Not supported. Its value is <b>NULL</b> .
table_type_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
table_type	character varying(128)	Not supported. Its value is <b>NULL</b> .
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .

### 12.3.166 MY\_TABLESPACES

MY\_TABLESPACES displays the description of tablespaces that store objects owned by users. By default, it is accessible to all users. This view exists in both

PG\_CATALOG and SYS schema. The logical structure features of GaussDB are different from those of the ORA database.

**Table 12-268** MY\_TABLESPACES columns

Name	Type	Description
tablespace_name	character varying(64)	Tablespace name
block_size	numeric	Not supported. Its value is <b>NULL</b> .
initial_extent	numeric	Not supported. Its value is <b>NULL</b> .
next_extent	numeric	Not supported. Its value is <b>NULL</b> .
min_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_extents	numeric	Not supported. Its value is <b>NULL</b> .
max_size	numeric	Not supported. Its value is <b>NULL</b> .
pct_increase	numeric	Not supported. Its value is <b>NULL</b> .
min_extlen	numeric	Not supported. Its value is <b>NULL</b> .
contents	character varying(9)	Not supported. Its value is <b>NULL</b> .
status	character varying(9)	Tablespace status. The default value is <b>ONLINE</b> .
logging	character varying(9)	Not supported. Its value is <b>NULL</b> .
force_logging	character varying(3)	Not supported. Its value is <b>NULL</b> .
extent_management	character varying(10)	Not supported. Its value is <b>NULL</b> .
allocation_type	character varying(9)	Not supported. Its value is <b>NULL</b> .
segment_space_management	character varying(6)	Not supported. Its value is <b>NULL</b> .
def_tab_compression	character varying(8)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
retention	character varying(11)	Not supported. Its value is <b>NULL</b> .
bigfile	character varying(3)	Not supported. Its value is <b>NULL</b> .
predicate_evaluation	character varying(7)	Not supported. Its value is <b>NULL</b> .
encrypted	character varying(3)	Not supported. Its value is <b>NULL</b> .
compress_for	character varying(30)	Not supported. Its value is <b>NULL</b> .
def_inmemory	character varying(8)	Not supported. Its value is <b>NULL</b> .
def_inmemory_priority	character varying(8)	Not supported. Its value is <b>NULL</b> .
def_inmemory_distribute	character varying(15)	Not supported. Its value is <b>NULL</b> .
def_inmemory_compression	character varying(17)	Not supported. Its value is <b>NULL</b> .
def_inmemory_duplicate	character varying(13)	Not supported. Its value is <b>NULL</b> .
shared	character varying(12)	Not supported. Its value is <b>NULL</b> .
def_index_compression	character varying(8)	Not supported. Its value is <b>NULL</b> .
index_compress_for	character varying(13)	Not supported. Its value is <b>NULL</b> .
def_cellmemory	character varying(14)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service	character varying(12)	Not supported. Its value is <b>NULL</b> .
def_inmemory_service_name	character varying(1000)	Not supported. Its value is <b>NULL</b> .
lost_write_protect	character varying(7)	Not supported. Its value is <b>NULL</b> .
chunk_tablespace	character varying(1)	Not supported. Its value is <b>NULL</b> .

## 12.3.167 MY\_TRIGGERS

MY\_TRIGGERS displays information about the triggers owned by the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-269** MY\_TRIGGERS columns

Name	Type	Description
owner	character varying(128)	Trigger owner.
trigger_name	character varying(64)	Trigger name.
trigger_type	character varying	Trigger types: <b>before statement</b> , <b>before each row</b> , <b>after statement</b> , <b>after each row</b> , and <b>instead of</b> .
triggering_event	character varying	Event that triggers a trigger: <b>update</b> , <b>insert</b> , <b>delete</b> , or <b>truncate</b> .
table_owner	character varying(64)	Owner of the table that defines a trigger.
base_object_type	character varying(18)	Basic object type of a trigger, which can be <b>table</b> or <b>view</b> .
table_name	character varying(64)	Name of the table or view that defines a trigger.
column_name	character varying(4000)	Not supported. Its value is <b>NULL</b> .
referencing_name	character varying(422)	Not supported. Its value is <b>referencing new as new old as old</b> .
when_clause	character varying(4000)	Content of <b>when</b> . <b>TRUE</b> must be evaluated as <b>TRIGGER_BODY</b> to execute.
status	character varying(64)	<ul style="list-style-type: none"><li>● <b>O</b>: The trigger is enabled in origin or local mode.</li><li>● <b>D</b>: The trigger is disabled.</li><li>● <b>R</b>: The trigger is enabled in replica mode.</li><li>● <b>A</b>: The trigger is always enabled.</li></ul>
description	character varying(4000)	Trigger description, which is used to rebuild a trigger creation statement.
action_type	character varying(11)	Action type of a trigger, which only supports <b>call</b> .
trigger_body	text	Statement executed when a trigger is triggered.

Name	Type	Description
crossedition	character varying(7)	Not supported. Its value is <b>NULL</b> .
before_statement	character varying(3)	Not supported. Its value is <b>NULL</b> .
before_row	character varying(3)	Not supported. Its value is <b>NULL</b> .
after_row	character varying(3)	Not supported. Its value is <b>NULL</b> .
after_statement	character varying(3)	Not supported. Its value is <b>NULL</b> .
instead_of_row	character varying(3)	Not supported. Its value is <b>NULL</b> .
fire_once	character varying(3)	Not supported. Its value is <b>NULL</b> .
apply_server_only	character varying(3)	Not supported. Its value is <b>NULL</b> .

## 12.3.168 MY\_TYPE\_ATTRS

MY\_TYPE\_ATTRS displays all types of attributes owned by the current user in the database. This view exists in both PG\_CATALOG and SYS schema and all users can access this view.

**Table 12-270** MY\_TYPE\_ATTRS columns

Name	Type	Description
type_name	character varying(128)	Data type name.
attr_name	character varying(128)	Attribute name.
attr_type_mod	character varying(7)	Type modifier of an attribute: <ul style="list-style-type: none"><li>• REF</li><li>• POINT</li></ul>
attr_type_owner	character varying(128)	Owner of an attribute type.
attr_type_name	character varying(128)	Name of an attribute type.
length	numeric	Length of the CHAR attribute, or the maximum length of the VARCHAR and character varying attribute
precision	numeric	Decimal precision of a number or DECIMAL attribute, or binary precision of a FLOAT attribute.

Name	Type	Description
scale	numeric	Decimal places for a numeric or DECIMAL attribute.
character_set_name	character varying(44)	Character set name of an attribute ( <b>Char_CS</b> or <b>NCHAR_CS</b> ).
attr_no	numeric	Syntax order number or location (not used as an ID number) of an attribute specified in the type specification or CREATE TYPE statement.
inherited	character varying(3)	Specifies whether an attribute is inherited from a supertype. <ul style="list-style-type: none"><li>• <b>YES</b>: It is inherited from a supertype.</li><li>• <b>NO</b>: It is not inherited from a supertype.</li></ul>

## 12.3.169 MY\_TYPES

MY\_TYPES describes all object types owned by the current user. All users can access this view. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-271** MY\_TYPES columns

Name	Type	Description
type_name	character varying(128)	Type name.
type_oid	raw	Type OID.
typecode	character varying(128)	Type code.
attributes	numeric	Number of attributes in a type.
methods	numeric	Not supported. Its value is <b>0</b> .
predefined	character varying(3)	Specifies whether the type is a predefined type.
incomplete	character varying(3)	Specifies whether the type is incomplete.
final	character varying(3)	Not supported. Its value is <b>NULL</b> .
instantiable	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
persistable	character varying(3)	Not supported. Its value is <b>NULL</b> .
supertype_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
supertype_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
local_attributes	numeric	Not supported. Its value is <b>NULL</b> .
local_methods	numeric	Not supported. Its value is <b>NULL</b> .
typeid	raw	Not supported. Its value is <b>NULL</b> .

### 12.3.170 MY\_VIEWS

MY\_VIEWS displays information about all views of the current user. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-272** MY\_VIEWS columns

Name	Type	Description
owner	character varying(64)	Owner of a view.
view_name	character varying(64)	View name.
text	text	Text in the view.
text_length	integer	Text length of the view.
TEXT_VC	character varying(4000)	View creation statement. This column may truncate the view text. The BEQUEATH clause will not appear as part of the <b>TEXT_VC</b> column in this view.
type_text_length	numeric	Not supported. Its value is <b>NULL</b> .
type_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .
oid_text_length	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
oid_text	character varying(4000)	Not supported. Its value is <b>NULL</b> .
view_type_owner	character varying(128)	Not supported. Its value is <b>NULL</b> .
view_type	character varying(128)	Not supported. Its value is <b>NULL</b> .
superview_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
editioning_view	character varying(1)	Not supported. Its value is <b>NULL</b> .
read_only	character varying(1)	Not supported. Its value is <b>NULL</b> .
container_data	character varying(1)	Not supported. Its value is <b>NULL</b> .
bequeath	character varying(12)	Not supported. Its value is <b>NULL</b> .
origin_con_id	character varying(256)	Not supported. Its value is <b>NULL</b> .
default_collation	character varying(100)	Not supported. Its value is <b>NULL</b> .
containers_default	character varying(3)	Not supported. Its value is <b>NULL</b> .
container_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link	character varying(3)	Not supported. Its value is <b>NULL</b> .
extended_data_link_map	character varying(3)	Not supported. Its value is <b>NULL</b> .
has_sensitive_column	character varying(3)	Not supported. Its value is <b>NULL</b> .
admit_null	character varying(3)	Not supported. Its value is <b>NULL</b> .
pdb_local_only	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.171 NLS\_DATABASE\_PARAMETERS

NLS\_DATABASE\_PARAMETERS lists the permanent NLS parameters of the database server. This view exists in both PG\_CATALOG and SYS schema. It is

accessible to all users. Due to different database kernels and parameter setting formats, the query results of the same parameters in this view may be obviously different from those in the ORA database.

**Table 12-273** NLS\_DATABASE\_PARAMETERS columns

Name	Type	Description
parameter	character varying(128)	Parameter name.
value	character varying(64)	Parameter value.

### 12.3.172 NLS\_INSTANCE\_PARAMETERS

NLS\_INSTANCE\_PARAMETERS lists the permanent NLS parameters of the database client. This view exists in both PG\_CATALOG and SYS schema. It is accessible to all users. Due to different database kernels and parameter setting formats, the query results of the same parameters in this view may be obviously different from those in the ORA database.

**Table 12-274** NLS\_INSTANCE\_PARAMETERS columns

Name	Type	Description
parameter	character varying(128)	Parameter name.
value	character varying(64)	Parameter value.

### 12.3.173 PG\_AVAILABLE\_EXTENSION\_VERSIONS

PG\_AVAILABLE\_EXTENSION\_VERSIONS displays extension versions of certain database features. This view is for internal use only. You are advised not to use it.

**Table 12-275** PG\_AVAILABLE\_EXTENSION\_VERSIONS columns

Name	Type	Description
name	name	Extension name.
version	text	Version name.
installed	Boolean	Specifies whether this extension version is installed. Otherwise, the value is <b>false</b> .
superuser	Boolean	Specifies whether only system administrators are allowed to install the extension. Otherwise, the value is <b>false</b> .

Name	Type	Description
relocatable	Boolean	Specifies whether the extension can be relocated to another schema. Otherwise, the value is <b>false</b> .
schema	name	Name of the schema that the extension must be installed into ( <b>NULL</b> if the extension is partially or fully relocatable).
requires	name[]	Names of prerequisite extensions ( <b>NULL</b> if none)
comment	text	Comment from the extension's control file.

## 12.3.174 PG\_AVAILABLE\_EXTENSIONS

PG\_AVAILABLE\_EXTENSIONS displays the extension information about certain database features. This view is for internal use only. You are advised not to use it.

Table 12-276 PG\_AVAILABLE\_EXTENSIONS columns

Name	Type	Description
name	name	Extension name.
default_version	text	Name of the default version ( <b>NULL</b> if none is specified).
installed_version	text	Currently installed version of the extension ( <b>NULL</b> if no version is installed).
comment	text	Comment from the extension's control file.

## 12.3.175 PG\_COMM\_DELAY

PG\_COMM\_DELAY displays the communication library delay status for a single DN.

Table 12-277 PG\_COMM\_DELAY columns

Name	Type	Description
node_name	text	Node name
remote_name	text	Name of the peer node
remote_host	text	IP address of the peer node
stream_num	integer	Number of logical stream connections used by the current physical connection

Name	Type	Description
min_delay	integer	Minimum delay of the current physical connection within 1 minute, in microsecond <b>NOTE</b> A negative result is invalid. Wait until the delay status is updated and query again.
average	integer	Average delay of the current physical connection within 1 minute, in microsecond
max_delay	integer	Maximum delay of the current physical connection within 1 minute, in microsecond

### 12.3.176 PG\_COMM\_RECV\_STREAM

**PG\_COMM\_RECV\_STREAM** displays the receiving stream status of all the communication libraries for a single DN.

**Table 12-278** PG\_COMM\_RECV\_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Status of the stream <ul style="list-style-type: none"> <li>• <b>UNKNOWN</b>: The logical connection status is unknown.</li> <li>• <b>READY</b>: The logical connection is ready.</li> <li>• <b>RUN</b>: The logical connection sends packets normally.</li> <li>• <b>HOLD</b>: The logical connection is waiting to send packets.</li> <li>• <b>CLOSED</b>: The logical connection is closed.</li> <li>• <b>TO_CLOSED</b>: The logical connection will be closed.</li> </ul>
query_id	bigint	<b>debug_query_id</b> corresponding to the stream

Name	Type	Description
pn_id	integer	<b>plan_node_id</b> of the query executed by the stream
send_smp	integer	<b>smpid</b> of the sender of the query executed by the stream
recv_smp	integer	<b>smpid</b> of the receiver of the query executed by the stream
recv_bytes	bigint	Total data volume received from the stream, in byte
time	bigint	Current lifecycle service duration of the stream, in ms
speed	bigint	Average receiving rate of the stream, in byte/s
quota	bigint	Current communication quota value of the stream, in byte
buff_use	bigint	Current size of the data cache of the stream, in byte

### 12.3.177 PG\_COMM\_SEND\_STREAM

**PG\_COMM\_SEND\_STREAM** displays the sending stream status of all the communication libraries for a single DN.

**Table 12-279** PG\_COMM\_SEND\_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream

Name	Type	Description
state	text	Status of the stream <ul style="list-style-type: none"><li>• <b>UNKNOWN</b>: The logical connection status is unknown.</li><li>• <b>READY</b>: The logical connection is ready.</li><li>• <b>RUN</b>: The logical connection sends packets normally.</li><li>• <b>HOLD</b>: The logical connection is waiting to send packets.</li><li>• <b>CLOSED</b>: The logical connection is closed.</li><li>• <b>TO_CLOSED</b>: The logical connection will be closed.</li></ul>
query_id	bigint	<b>debug_query_id</b> corresponding to the stream
pn_id	integer	<b>plan_node_id</b> of the query executed by the stream
send_smp	integer	<b>smpid</b> of the sender of the query executed by the stream
rcv_smp	integer	<b>smpid</b> of the receiver of the query executed by the stream
send_bytes	bigint	Total data volume sent by the stream, in byte
time	bigint	Current lifecycle service duration of the stream, in ms
speed	bigint	Average sending rate of the stream, in byte/s
quota	bigint	Current communication quota value of the stream, in byte
wait_quota	bigint	Extra time generated when the stream waits the quota value, in ms

## 12.3.178 PG\_COMM\_STATUS

**PG\_COMM\_STATUS** displays the communication library status for a single DN.

**Table 12-280** PG\_COMM\_STATUS columns

Name	Type	Description
node_name	text	Node name
rxpck_rate	integer	Receiving rate of the communication library on the node, in byte/s

Name	Type	Description
txpck_rate	integer	Sending rate of the communication library on the node, in byte/s
rxkbyte_rate	bigint	Receiving rate of the communication library on the node, in kbyte/s
txkbyte_rate	bigint	Sending rate of the communication library on the node, in kbyte/s
buffer	bigint	Size of the buffer of the Cmailbox
memkbyte_libcomm	bigint	Communication memory size of the <b>libcomm</b> process, in bytes
memkbyte_libpq	bigint	Communication memory size of the <b>libpq</b> process, in bytes
used_pm	integer	Real-time usage of the <b>postmaster</b> thread
used_sflow	integer	Real-time usage of the <b>gs_sender_flow_controller</b> thread
used_rflow	integer	Real-time usage of the <b>gs_receiver_flow_controller</b> thread
used_rloop	integer	Highest real-time usage among multiple <b>gs_receivers_loop</b> threads
stream	integer	Total number of used logical connections.

### 12.3.179 PG\_CONTROL\_GROUP\_CONFIG

PG\_CONTROL\_GROUP\_CONFIG displays Cgroup configuration information in the system. Only the user with sysadmin permission can query this view.

**Table 12-281** PG\_CONTROL\_GROUP\_CONFIG columns

Name	Type	Description
pg_control_group_config	text	Configuration information of the Cgroup

### 12.3.180 PG\_CURSORS

PG\_CURSORS displays cursors that are currently available.

**Table 12-282** PG\_CURSORS columns

Name	Type	Description
name	text	Cursor name.
statement	text	Query statement when the cursor is declared to change.
is_holdable	Boolean	Specifies whether the cursor is holdable (that is, whether the cursor can be accessed after the transaction that declared the cursor has committed). If it is, the value is <b>TRUE</b> . Otherwise, the value is <b>FALSE</b> .
is_binary	Boolean	Specifies whether the cursor is declared BINARY. If it is, the value is <b>TRUE</b> . Otherwise, the value is <b>FALSE</b> .
is_scrollable	Boolean	Specifies whether the cursor is scrollable (it allows rows to be retrieved in a nonsequential manner). If it is, the value is <b>TRUE</b> . Otherwise, the value is <b>FALSE</b> .
creation_time	timestamp with time zone	Timestamp at which the cursor is declared.

## 12.3.181 PG\_EXT\_STATS

PG\_EXT\_STATS displays extended statistics stored in [PG\\_STATISTIC\\_EXT](#). The extension statistics means multiple columns of statistics.

**Table 12-283** PG\_EXT\_STATS columns

Name	Type	Reference	Description
schemaname	name	<b>nspname</b> in <a href="#">PG_NAMESPACE</a>	Name of the schema that contains a table.
tablename	name	<b>relname</b> in <a href="#">PG_CLASS</a>	Table name.
attname	int2vector	<b>stakey</b> in <a href="#">PG_STATISTIC_EXT</a>	Columns to be combined for collecting statistics
inherited	Boolean	-	Table inheritance is not supported currently. The value of this column is <b>false</b> .
null_frac	real	-	Percentage of column combinations that are null to all records.

Name	Type	Reference	Description
avg_width	integer	-	Average width of column combinations, in bytes.
n_distinct	real	-	<ul style="list-style-type: none"> <li>Estimated number of distinct values in a column combination if the value is greater than 0.</li> <li>Negative number obtained by multiplying the result calculated by dividing the number of distinct values by the number of rows by -1 if the value is less than 0. For example, -1 indicates that the number of distinct values is the same as the number of rows for a column combination.               <ol style="list-style-type: none"> <li>The negated form is used when <b>ANALYZE</b> believes that the number of distinct values is likely to increase as the table grows.</li> <li>The positive form is used when the column seems to have a fixed number of possible values.</li> </ol> </li> <li>The number of distinct values is unknown if the value is 0.</li> </ul>
n_dndistinct	real	-	<p>Number of not-null distinct values in the <b>dn1</b> column combination.</p> <ul style="list-style-type: none"> <li>Exact number of distinct values if the value is greater than 0.</li> <li>Negative number obtained by multiplying the result calculated by dividing the number of distinct values by the number of rows by -1 if the value is less than 0. For example, if a value in a column combination appears twice in average, <b>n_dndistinct</b> equals -0.5.</li> <li>The number of distinct values is unknown if the value is 0.</li> </ul>

Name	Type	Reference	Description
most_common_vals	anyarray	-	List of the most common values in a column combination. If this combination does not have the most common values, this column will be <b>NULL</b> . None of the most common values in the column is <b>NULL</b> .
most_common_freqs	real[]	-	List of the frequencies of the most common values in a column combination. The frequencies are obtained by dividing the number of occurrences of each value by the number of rows. If the value of <b>most_common_vals</b> is <b>NULL</b> , the value of this column is <b>NULL</b> .
most_common_vals_null	anyarray	-	List of the most common values in a column combination. If this combination does not have the most common values, this column will be <b>NULL</b> . At least one of the common values in the column is <b>NULL</b> .
most_common_freqs_null	real[]	-	List of the frequencies of the most common values in a column combination. The frequencies are obtained by dividing the number of occurrences of each value by the number of rows. If the value of <b>most_common_vals_null</b> is <b>NULL</b> , the value of this column is <b>NULL</b> .
histogram_bounds	anyarray	-	Boundary value list of the histogram.

### 12.3.182 PG\_GET\_INVALID\_BACKENDS

The PG\_GET\_INVALID\_BACKENDS view displays information about current standby DN backend threads connected to the CN. Only system administrators and monitor administrators can access this view.

**Table 12-284** PG\_GET\_INVALID\_BACKENDS columns

Name	Type	Description
pid	bigint	Thread ID
node_name	text	Node information connected to the backend thread
dbname	name	Name of the connected database
backend_start	timestamp with time zone	Backend thread startup time
query	text	Query statement executed by the backend thread

### 12.3.183 PG\_GET\_SENDERS\_CATCHUP\_TIME

PG\_GET\_SENDERS\_CATCHUP\_TIME displays catchup information of the currently active primary/standby instance sender thread on the DN.

**Table 12-285** PG\_GET\_SENDERS\_CATCHUP\_TIME columns

Name	Type	Description
pid	bigint	Current sender thread ID.
lwpid	integer	Current sender lwpid.
local_role	text	Local role.
peer_role	text	Peer role.
state	text	Current sender's replication status <ul style="list-style-type: none"> <li>• <b>Startup</b>: startup state.</li> <li>• <b>Backup</b>: backup state.</li> <li>• <b>Catchup</b>: catch-up state, which indicates that the standby node is catching up with the primary node.</li> <li>• <b>Streaming</b>: streaming replication state. When the standby node catches up with the primary node, the replication remains in this state.</li> </ul>
type	text	Current sender type <ul style="list-style-type: none"> <li>• <b>Wal</b>: type that logs are written in advance.</li> <li>• <b>Data</b>: data type.</li> </ul>

Name	Type	Description
catchup_start	timestamp with time zone	Startup time of a catchup task.
catchup_end	timestamp with time zone	End time of the catchup task.

## 12.3.184 PG\_GROUP

PG\_GROUP displays the database role authentication and the relationship between group members.

**Table 12-286** PG\_GROUP columns

Name	Type	Description
groname	name	Group name
grosysid	oid	Group ID
grolist	oid[]	Array containing the IDs of all roles in the group.

## 12.3.185 PG\_INDEXES

PG\_INDEXES displays information about each index in the database.

**Table 12-287** PG\_INDEXES columns

Name	Type	Reference	Description
schemaname	name	<b>nspname</b> in <b>PG_NAMESPACE</b>	Name of the schema that contains tables and indexes.
tablename	name	<b>relname</b> in <b>PG_CLASS</b>	Name of the table where the index is located.
indexname	name	<b>relname</b> in <b>PG_CLASS</b>	Index name
tablespace	name	<b>nspname</b> in <b>PG_TABLESPACE</b>	Name of the tablespace that contains the index.

Name	Type	Reference	Description
indexdef	text	-	Index definition (a reconstructed <b>CREATE INDEX</b> command).

## 12.3.186 PG\_LOCKS

PG\_LOCKS displays information about locks held by open transactions.

**Table 12-288** PG\_LOCKS columns

Name	Type	Reference	Description
locktype	text	-	Type of the locked object. The value can be <b>relation</b> , <b>extend</b> , <b>page</b> , <b>tuple</b> , <b>transactionid</b> , <b>virtualxid</b> , <b>object</b> , <b>userlock</b> , or <b>advisory</b> .
database	oid	<b>oid</b> in <b>PG_DATABASE</b>	OID of the database in which the locked target exists. <ul style="list-style-type: none"> <li>The OID is <b>0</b> if the target is a shared object.</li> <li>The OID is <b>NULL</b> if the locked object is a transaction.</li> </ul>
relation	oid	<b>oid</b> in <b>PG_CLASS</b>	OID of the relationship targeted by the lock. The value is <b>NULL</b> if the object is not a relationship or part of a relationship.
page	integer	-	Page number targeted by the lock within the relationship ( <b>NULL</b> if the object is not a relation page or row page).
tuple	smallint	-	Row number targeted by the lock within the page ( <b>NULL</b> if the object is not a row).
bucket	integer	-	Hash bucket ID.
virtualxid	text	-	ID of the virtual transaction. The value is <b>NULL</b> if the object is not a virtual transaction.
transactionid	xid	-	ID of the transaction. The value is <b>NULL</b> if the object is not a transaction.

Name	Type	Reference	Description
classid	oid	<b>oid</b> in <b>PG_CLASS</b>	OID of the system catalog that contains the object ( <b>NULL</b> if the object is not a general database object).
objid	oid	-	OID of the locked object within its system catalog. The value is <b>NULL</b> if the object is not a general database object.
objsubid	smallint	-	Column number for a column in the table ( <b>0</b> if the target is of other object type and <b>NULL</b> if the target is not a general database object).
virtualtrans action	text	-	Virtual ID of the virtual transaction holding or awaiting this lock.
pid	bigint	-	Logical ID of the server thread holding or awaiting this lock ( <b>NULL</b> if the lock is held by a prepared transaction).
sessionid	bigint	-	ID of the session that holds or waits for the lock.
mode	text	-	Lock mode held or desired by this thread.
granted	Boolean	-	<ul style="list-style-type: none"><li>The value is <b>TRUE</b> if the lock is a held lock.</li><li>The value is <b>FALSE</b> if the lock is an awaited lock.</li></ul>
fastpath	Boolean	-	The value is <b>TRUE</b> if the lock is obtained through <b>fast-path</b> , and is <b>FALSE</b> if the lock is obtained through the main lock table.
locktag	text	-	Lock information that the session waits for. It can be parsed using the locktag_decode() function.
global_sessionid	text	-	Global session ID.

### 12.3.187 PG\_NODE\_ENV

PG\_NODE\_ENV displays the environment variable information of the current node. This view can be viewed only by the users with monitor admin and sysadmin permission.

**Table 12-289** PG\_NODE\_ENV columns

Name	Type	Description
node_name	text	Current node name.
host	text	Host name of the node.
process	integer	Number of the node process.
port	integer	Port ID of the node.
installpath	text	Installation directory of the node.
datapath	text	Data directory of the node.
log_directory	text	Log directory of the node.

## 12.3.188 PG\_OS\_THREADS

**PG\_OS\_THREADS** provides status information about all the threads under the current node.

**Table 12-290** PG\_OS\_THREADS columns

Name	Type	Description
node_name	text	Current node name
pid	bigint	PID of the thread running under the current node process
lwpid	integer	Lightweight thread ID corresponding to the PID
thread_name	text	Thread name corresponding to the PID
creation_time	timestamp with time zone	Thread creation time corresponding to the PID

## 12.3.189 PG\_POOLER\_STATUS

**PG\_POOLER\_STATUS** queries the cache connection status in the pooler. This view can only query on the CN, and displays the connection cache information about the pooler module.

**Table 12-291** PG\_POOLER\_STATUS columns

Name	Type	Description
database	text	Database name.

Name	Type	Description
user_name	text	Username.
tid	bigint	In non-thread pool logic, this parameter indicates the ID of the thread connected to the CN. In thread pool logic, this parameter indicates the ID of the session connected to the CN.
node_oid	bigint	OID of the node connected.
node_name	name	Name of the node connected.
in_use	Boolean	Specifies whether the connection is currently used. <ul style="list-style-type: none"><li>• <b>t</b> (true): The connection is in use.</li><li>• <b>f</b> (false): The connection is not in use.</li></ul>
node_port	integer	Port number of the connected instance node.
fdsock	bigint	Peer socket.
remote_pid	bigint	Peer thread ID.
session_params	text	GUC session parameter delivered by the connection.
used_count	bigint	Number of reuse times of a connection.
idx	bigint	Logical connection ID of the connected instance node.
streamid	bigint	Stream ID corresponding to each logical connection.

## 12.3.190 PG\_PREPARED\_STATEMENTS

PG\_PREPARED\_STATEMENTS displays information about all prepared statements that are available in the current session.

**Table 12-292** PG\_PREPARED\_STATEMENTS columns

Name	Type	Description
name	text	Identifier of a prepared statement.

Name	Type	Description
statement	text	Query string for creating this prepared statement. For prepared statements created through SQL, this is the PREPARE statement committed by the client. For prepared statements created through the frontend/backend protocol, this is the text of the prepared statement itself.
prepare_time	timestamp with time zone	Timestamp when the prepared statement is created.
parameter_types	regtype[]	Expected parameter types for the prepared statement in the form of an array of <b>regtype</b> . The OID corresponding to an element of this array can be obtained by converting the <b>regtype</b> value to an OID value.
from_sql	Boolean	<ul style="list-style-type: none"> <li>Its value is <b>true</b> if the prepared statement was created through the PREPARE statement</li> <li>Its value is <b>false</b> if the statement was prepared through the frontend/backend protocol</li> </ul>

### 12.3.191 PG\_PREPARED\_XACTS

PG\_PREPARED\_XACTS displays information about transactions that are currently prepared for two-phase commit.

**Table 12-293** PG\_PREPARED\_XACTS columns

Name	Type	Reference	Description
transaction	xid	-	Numeric identifier of the prepared transaction.
gid	text	-	Global identifier of the prepared transaction.
prepared	timestamp with time zone	-	Time at which the transaction is prepared for commit
owner	name	<a href="#">PG_AUTHID</a> .rolname	Name of the user who executes the transaction.
database	name	<a href="#">PG_DATABASE</a> .datname	Name of the database that executes the transaction.

## 12.3.192 PG\_REPLICATION\_SLOTS

PG\_REPLICATION\_SLOTS displays replication slot information.

**Table 12-294** PG\_REPLICATION\_SLOTS columns

Name	Type	Description
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.
slot_type	text	Replication slot type. <ul style="list-style-type: none"><li>• <b>physical</b>: physical replication slot.</li><li>• <b>logical</b>: logical replication slot.</li></ul>
datoid	oid	OID of the database where the replication slot resides.
database	name	Name of the database where the replication slot resides.
active	Boolean	Determines whether the replication slot is activated. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes.</li><li>• <b>f</b> (false): no.</li></ul>
xmin	xid	XID of the earliest transaction that the database must reserve for the replication slot.
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.
dummy_standby	Boolean	Reserved parameter.
confirmed_flush	text	Dedicated logical replication slot. The client confirms the location of the received log.
confirmed_csn	xid	Dedicated logical replication slot. The client confirms the CSN corresponding to the last transaction in the received log.

Example:

```
-- Perform query on the DN.
gaussdb=# SELECT * FROM pg_replication_slots;
slot_name | plugin | slot_type | datoid | database | active | xmin | catalog_xmin | restart_lsn |
dummy_standby | confirmed_flush | confirmed_csn
```



Name	Type	Description
polycmd	text	SQL operations affected by the row-level security policy.
policyqual	text	Expression of a row-level security policy.

## 12.3.194 PG\_ROLES

PG\_ROLES displays information about database roles. Initialization users and users with the sysadmin or createrole attribute can view information about all roles. Other users can view only their own information.

**Table 12-296** PG\_ROLES columns

Name	Type	Reference	Description
rolname	name	N/A	Role name.
rolsuper	Boolean	N/A	Specifies whether a role is the initial system administrator with the highest permissions. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolinherit	Boolean	N/A	Specifies whether the role inherits the permissions for this type of roles. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolcreaterole	Boolean	N/A	Specifies whether the role can create other roles. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolcreatedb	Boolean	N/A	Specifies whether the role can create databases. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolcatupdate	Boolean	N/A	Specifies whether the role can update system catalogs directly. Only the initial system administrator whose <b>usesysid</b> is <b>10</b> has this permission. This permission is unavailable for other users. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>

Name	Type	Reference	Description
rolcanlogin	Boolean	N/A	Specifies whether the role can log in to the database. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolreplication	Boolean	N/A	Specifies whether the role can be replicated. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolauditadmin	Boolean	N/A	Specifies whether the role is an audit administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolsystemadmin	Boolean	N/A	Specifies whether the role is a system administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolconnlimit	integer	N/A	Maximum number of concurrent connections that this role can initiate if this role can log in. The value <b>-1</b> indicates no limit.
rolpassword	text	N/A	Encrypted user password. The value is displayed as <b>*****</b> .
rolvalidbegin	timestamp with time zone	N/A	Start time for account validity ( <b>NULL</b> if start time is not specified).
rolvaliduntil	timestamp with time zone	N/A	End time for account validity ( <b>NULL</b> if end time is not specified).
rolrespool	name	N/A	Resource pool that a user can use
rolparentid	oid	<a href="#">PG_AUTHID.rolparentid</a>	OID of a group user to which the user belongs
roltabspace	text	N/A	Storage space of the user permanent table, in KB.
rolconfig	text[]	<a href="#">PG_DB_ROLE_SETTING.setconfig</a>	Default value of runtime configuration items.
oid	oid	<b>oid</b> in <a href="#">PG_AUTHID</a>	Role ID

Name	Type	Reference	Description
roluseft	Boolean	<a href="#">PG_AUTHID.roluseft</a>	Specifies whether the role can perform operations on foreign tables <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolkind	"char"	N/A	Role type <ul style="list-style-type: none"> <li>• <b>n</b>: common user, that is, non-permanent user</li> <li>• <b>p</b>: permanent user.</li> </ul>
roltemp space	text	N/A	Storage space of the user temporary table, in KB.
rolspill space	text	N/A	Operator disk spill space of the user, in KB.
rolmonitor administrator	Boolean	N/A	Specifies whether the role is a monitor administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
roloperator administrator	Boolean	N/A	Specifies whether the role is an O&M administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
rolpolicy administrator	Boolean	N/A	Specifies whether the role is a security policy administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>

## 12.3.195 PG\_RULES

PG\_RULES is used to query useful information about rewriting rules.

**Table 12-297** PG\_RULES columns

Name	Type	Description
schemaname	name	Name of the schema that contains a table.
tablename	name	Name of the table to which the rule applies.
rulename	name	Name of a rule.

Name	Type	Description
definition	text	Rule definition (reconstructed by the CREATE statement).

## 12.3.196 PG\_RUNNING\_XACTS

PG\_RUNNING\_XACTS displays information about running transactions on the current node.

**Table 12-298** PG\_RUNNING\_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM.
gxid	xid	Transaction ID.
state	tinyint	Transaction status. <ul style="list-style-type: none"><li>• <b>3</b>: prepared.</li><li>• <b>0</b>: starting.</li></ul>
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	Boolean	Specifies whether the current transaction is lazy vacuum. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes.</li><li>• <b>f</b> (false): no.</li></ul>
timeline	bigint	Number of database restarts.
prepare_xid	xid	ID of the transaction in the prepared state; otherwise, the value is <b>0</b> .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Minimum CSN of a local active transaction.

## 12.3.197 PG\_SECLABELS

PG\_SECLABELS provides information about security labels.

**Table 12-299** PG\_SECLABELS columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to.
classoid	oid	<b>oid</b> in <a href="#">PG_CLASS</a>	OID of the system catalog where the object to which the security label belongs is located.
objsubid	integer	-	Column number for the security label on a table column ( <b>objoid</b> and <b>classoid</b> refer to the table itself). The value is <b>0</b> for all other object types.
objtype	text	-	Type of the object to which the label belongs, in text format. Example: <ul style="list-style-type: none"> <li>• <b>table</b>: table type.</li> <li>• <b>column</b>: column type.</li> </ul>
objnamespace	oid	<b>oid</b> in <a href="#">PG_NAMESPACE</a>	OID of the namespace for the object, if applicable; otherwise, <b>NULL</b> .
objname	text	-	Name of the object to which the label belongs, in text format.
provider	text	<b>provider</b> in <a href="#">PG_SECLABEL</a>	Provider of the label.
label	text	<b>label</b> in <a href="#">PG_SECLABEL</a>	Security label name.

## 12.3.198 PG\_SETTINGS

PG\_SETTINGS displays information about parameters of the running database.

**Table 12-300** PG\_SETTINGS columns

Name	Type	Description
name	text	Parameter name.
setting	text	Current parameter value.
unit	text	Unit of the parameter.
category	text	Logical group of the parameter.

Name	Type	Description
short_desc	text	Brief description of the parameter.
extra_desc	text	Detailed description of the parameter.
context	text	Context required to set the parameter value, including <b>internal</b> , <b>postmaster</b> , <b>siguhp</b> , <b>backend</b> , <b>superuser</b> , and <b>user</b> .
vartype	text	Parameter type, including <b>bool</b> , <b>enum</b> , <b>integer</b> , <b>real</b> , or <b>string</b> .
source	text	Method of assigning the parameter value.
min_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is <b>null</b> .
max_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is <b>null</b> .
enumvals	text[]	Valid values of an enum-type parameter. If the parameter type is not enum, the value of this column is <b>null</b> .
boot_val	text	Default parameter value used upon the database startup.
reset_val	text	Default parameter value used upon the database reset.
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is <b>null</b> .
sourceline	integer	Row number of the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is <b>null</b> .

### 12.3.199 PG\_SHADOW

PG\_SHADOW displays the attributes of all roles marked with rolcanlogin in [PG\\_AUTHID](#). Only the system administrator can access this system view.

The information in this view is basically the same as that of the view in [PG\\_USER](#). The difference is that passwords are sensitive and displayed as **\*\*\*\*\*** in the latter.

**Table 12-301** PG\_SHADOW columns

Name	Type	Reference	Description
username	name	<b>rolname</b> in <b>PG_AUTHID</b>	Username.
usesysid	oid	<b>oid</b> in <b>PG_AUTHID</b>	ID of this user.
usecreatedb	Boolean	-	Specifies whether a user has the permissions to create databases. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
usesuper	Boolean	-	Specifies whether the user is a system administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
usecatupd	Boolean	-	Specifies whether the user can update a view. Even the system administrator cannot do this unless this column is <b>true</b> . <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
userepl	Boolean	-	Specifies whether the user has the permissions to duplicate data streams. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
passwd	text	<b>rolpassword</b> in <b>PG_AUTHID</b>	Password ciphertext ( <b>NULL</b> if there is no password).
valbegin	timestamp with time zone	-	Start time for account validity ( <b>NULL</b> if start time is not specified).
valuntil	timestamp with time zone	-	End time for account validity ( <b>NULL</b> if end time is not specified).
respool	name	-	Resource pool where the user is in.
parent	oid	-	Parent user OID.
spacelimit	text	-	Storage space of the permanent table, in KB.

Name	Type	Reference	Description
useconfig	text[]	<b>setconfig</b> in <b>PG_DB_ROLE_SETTING</b>	Default value of runtime configuration items.
tempspacelimit	text	-	Storage space of the temporary table, in KB.
spillspacelimit	text	-	Operator disk spill space, in KB.
usemonitoradmin	Boolean	-	Specifies whether the user is a monitor administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
useoperatoradmin	Boolean	-	Specifies whether the user is an O&M administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
usepolicyadmin	Boolean	-	Specifies whether the user is a security policy administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>

## 12.3.200 PG\_SHARED\_MEMORY\_DETAIL

PG\_SHARED\_MEMORY\_DETAIL displays usage information about all the shared memory contexts.

**Table 12-302** PG\_SHARED\_MEMORY\_DETAIL columns

Name	Type	Description
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.
parent	text	Name of the parent memory context.
totalsize	bigint	Total size of the shared memory, in bytes.
freesize	bigint	Remaining size of the shared memory, in bytes.
usedsize	bigint	Used size of the shared memory, in bytes.

## 12.3.201 PG\_STAT\_ACTIVITY

PG\_STAT\_ACTIVITY shows information about the current user's queries. The columns save information about the last query.

**Table 12-303** PG\_STAT\_ACTIVITY columns

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend.
datname	name	Name of the database that the user session connects to in the backend.
pid	bigint	Backend thread ID.
sessionid	bigint	Session ID.
usesysid	oid	OID of the user logged in to the backend.
username	name	Name of the user logged in to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is <b>null</b> , it indicates either the client is connected via a Unix socket on the server or this is an internal thread, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend ( <b>-1</b> if a Unix socket is used).
backend_start	timestamp with time zone	Time when this session was started, that is, when the client connected to the server.
xact_start	timestamp with time zone	Time when the current active transaction was started ( <b>null</b> if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the <b>query_start</b> column.

Name	Type	Description
query_start	timestamp with time zone	Time when the current active query was started, or time when the last query was started if the value of <b>state</b> is not <b>active</b> . For a stored procedure or function, the first query time is displayed and does not change with the running of statements in the stored procedure.
state_change	timestamp with time zone	Time when <b>state</b> was last modified.
waiting	Boolean	Specifies whether the backend is currently waiting for a lock. If the backend is currently waiting for a lock, the value is <b>true</b> . Otherwise, the value is <b>false</b> .
enqueue	text	Queuing status of a statement. Its value can be: <ul style="list-style-type: none"> <li>• <b>waiting in queue</b>: The statement is in the queue.</li> <li>• <b>Empty</b>: The statement is running.</li> </ul>

Name	Type	Description
state	text	<p>Overall status of this backend. Its value can be:</p> <ul style="list-style-type: none"> <li>● <b>active</b>: The backend is executing a query.</li> <li>● <b>idle</b>: The backend is waiting for a new client command.</li> <li>● <b>idle in transaction</b>: The backend is in a transaction, but there is no statement being executed in the transaction.</li> <li>● <b>idle in transaction (aborted)</b>: The backend is in a transaction, but there are statements failed in the transaction.</li> <li>● <b>fastpath function call</b>: The backend is executing a fast-path function.</li> <li>● <b>disabled</b>: This state is reported if <b>track_activities</b> is disabled in this backend.</li> </ul> <p><b>NOTE</b> Common users can view their own session status only. The state information of other accounts is empty. For example, after user <b>judy</b> is connected to the database, the state information of user <b>joe</b> and the initial user <b>omm</b> in <code>pg_stat_activity</code> is empty.</p> <pre>SELECT datname, username, usesysid, state,pid FROM pg_stat_activity;  datname   username   usesysid   state    pid -----+-----+-----+-----+-----  testdb   omm        10               139968752121616  testdb   omm        10               139968903116560  db_tpcds   judy      16398     active   139968391403280  testdb   omm        10               139968643069712  testdb   omm        10               139968680818448  testdb   joe       16390            139968563377936 (6 rows)</pre>
resource_pool	name	Resource pool used by the user.
query_id	bigint	ID of a query.

Name	Type	Description
query	text	Latest query at the backend. If <b>state</b> is <b>active</b> , this column shows the ongoing query. In all other states, it shows the last query that was executed.
connection_info	text	A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database. For details, see the GUC parameter <b>connection_info</b> .
global_sessionid	text	Global session ID.
unique_sql_id	bigint	Unique SQL statement ID.
trace_id	text	Driver-specific trace ID, which is associated with an application request.
top_xid	xid	Top-level transaction ID of a transaction.
current_xid	xid	Current transaction ID of a transaction.
xlog_quantity	bigint	Amount of Xlogs currently used by a transaction, in bytes.

## 12.3.202 PG\_STAT\_ALL\_INDEXES

PG\_STAT\_ALL\_INDEXES is used to query statistics about accesses to a specific index by querying each index row in the current database.

Indexes can be used via either simple index scans or bitmap index scans. In a bitmap scan the output of several indexes can be combined via AND or OR rules, so it is difficult to associate individual heap row fetches with specific indexes when a bitmap scan is used. Therefore, each bitmap scan increments the **pg\_stat\_all\_indexes.idx\_tup\_read** count(s) for the index(es) it uses, and it increments the **pg\_stat\_all\_tables.idx\_tup\_fetch** count for the table, but it does not affect **pg\_stat\_all\_indexes.idx\_tup\_fetch**.

**Table 12-304** PG\_STAT\_ALL\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for.
indexrelid	oid	OID of the index.

Name	Type	Description
schemaname	name	Name of the schema that the index is in.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans initiated on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 12.3.203 PG\_STAT\_ALL\_TABLES

PG\_STAT\_ALL\_TABLES is used to query one row for each table in the current database (including TOAST tables), showing statistics about a specific table.

**Table 12-305** PG\_STAT\_ALL\_TABLES columns

Name	Type	Description
relid	oid	OID of a table.
schemaname	name	Name of the schema that the table is in.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.

Name	Type	Description
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time when the table was cleared.
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times the table is cleared.
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times the table has been manually analyzed.
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon.
last_data_changed	timestamp with time zone	Last time when the data in the table changes (modifications that cause data changes on tables (insert/update/delete/truncate) and partitioned or subpartitioned tables (exchange/truncate/drop)). Data in this column is recorded only in the local database primary node.

### 12.3.204 PG\_STAT\_BAD\_BLOCK

PG\_STAT\_BAD\_BLOCK displays statistics about page verification failures after a node is started.

**Table 12-306** PG\_STAT\_BAD\_BLOCK columns

Name	Type	Description
nodename	text	Node name.
databaseid	integer	OID of a database.
tablespaceid	integer	Tablespace OID.
relfilenode	integer	File object ID.
bucketid	smallint	ID of the bucket for consistent hashing.
forknum	integer	File type. The values are as follows: <b>0</b> : main data file. <b>1</b> : FSM file. <b>2</b> : VM file. <b>3</b> : BCM file.
error_count	integer	Number of verification failures.
first_time	timestamp with time zone	Time of the first verification failure.
last_time	timestamp with time zone	Time of the latest verification failure.

### 12.3.205 PG\_STAT\_BGWRITER

PG\_STAT\_BGWRITER displays statistics about the activity of the background writer process.

**Table 12-307** PG\_STAT\_BGWRITER columns

Name	Type	Description
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed.
checkpoints_req	bigint	Number of checkpoints that have been performed actively.
checkpoint_write_time	double precision	Total time spent in the checkpoint processing portion when writing files to disk, in milliseconds.

Name	Type	Description
checkpoint_sync_time	double precision	Total time spent in the checkpoint processing portion when synchronizing files to disk, in milliseconds.
buffers_checkpoint	bigint	Number of buffers written by checkpoints.
buffers_clean	bigint	Number of buffers written by the background writer process.
maxwritten_clean	bigint	Number of times that cleanup scanning stops because the background writer process writes too many buffers.
buffers_backend	bigint	Number of buffers written directly by the backend.
buffers_backend_fsync	bigint	Number of times that the backend calls fsync (usually, even if the backend executes these write actions, the background writer process processes them again).
buffers_alloc	bigint	Number of buffers allocated.
stats_reset	timestamp with time zone	Time when these statistics were last reset.

## 12.3.206 PG\_STAT\_DATABASE

PG\_STAT\_DATABASE displays statistics about each database in the cluster.

**Table 12-308** PG\_STAT\_DATABASE columns

Name	Type	Description
datid	oid	OID of a database
datname	name	Database name.
numbackends	integer	Number of backends currently connected to this database. This is the only column in the view that returns the status value. Other columns return the accumulated value since the last reset.
xact_commit	bigint	Number of transactions in this database that have been committed.

Name	Type	Description
xact_rollback	bigint	Number of transactions in this database that have been rolled back.
blks_read	bigint	Number of disk blocks read in this database.
blks_hit	bigint	Number of times that disk blocks have been found in the buffer cache and therefore the disk blocks do not need to be read (only those found in the buffer cache are counted, excluding those found in the file system cache of the OS).
tup_returned	bigint	Number of rows returned by queries in this database
tup_fetched	bigint	Number of rows fetched by queries in this database
tup_inserted	bigint	Number of rows inserted by queries in this database
tup_updated	bigint	Number of rows updated by queries in this database
tup_deleted	bigint	Number of rows deleted by queries in this database
conflicts	bigint	Number of queries canceled due to database recovery conflicts (conflicts occurring only on the standby node). For details, see <a href="#">PG_STAT_DATABASE_CONFLICTS</a> .
temp_files	bigint	Number of all temporary files created by queries in this database, regardless of why the temporary files are created (such as sorting or hashing) or how the <b>log_temp_files</b> parameter is set.
temp_bytes	bigint	Total amount of data written to all temporary files by queries in this database, regardless of why the temporary files are created or how the <b>log_temp_files</b> parameter is set.
deadlocks	bigint	Number of deadlocks detected in the database.
blk_read_time	double precision	Time spent in reading data file blocks by backends in this database (unit: ms).
blk_write_time	double precision	Time spent in writing data file blocks by backends in this database (unit: ms).

Name	Type	Description
stats_reset	timestamp with time zone	Time when the status statistics are reset.

## 12.3.207 PG\_STAT\_DATABASE\_CONFLICTS

PG\_STAT\_DATABASE\_CONFLICTS displays statistics about database conflicts.

**Table 12-309** PG\_STAT\_DATABASE\_CONFLICTS columns

Name	Type	Description
datid	oid	ID of a database
datname	name	Database name
confl_tablespace	bigint	Number of conflicting tablespaces
confl_lock	bigint	Number of conflicting locks
confl_snapshot	bigint	Number conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

## 12.3.208 PG\_STAT\_REPLICATION

PG\_STAT\_REPLICATION displays information about the log synchronization thread, such as the locations where the sender sends logs and where the receiver receives logs.

**Table 12-310** PG\_STAT\_REPLICATION columns

Name	Type	Description
pid	bigint	PID of the thread.
usesysid	oid	User system ID.
username	name	Username.
application_name	text	Program name.
client_addr	inet	Client address.
client_hostname	text	Client name.
client_port	integer	Port of the client.

Name	Type	Description
backend_start	timestamp with time zone	Start time of the program.
state	text	Status of the log synchronization thread: <ul style="list-style-type: none"> <li>● <b>startup</b>: The thread is being started.</li> <li>● <b>catchup</b>: The thread is establishing a connection between the standby node and the primary node.</li> <li>● <b>streaming</b>: The thread has established a connection between the standby node and the primary node and is replicating data streams.</li> <li>● <b>backup</b>: The thread is sending a backup.</li> <li>● <b>stopping</b>: The thread is being stopped.</li> </ul>
sender_sent_location	text	Location where the sender sends logs.
receiver_write_location	text	Location where the receiver writes logs.
receiver_flush_location	text	Location where the receive end flushes logs.
receiver_replay_location	text	Location where the receive end replays logs.
sync_priority	integer	Priority of synchronous duplication ( <b>0</b> indicates asynchronization).

Name	Type	Description
sync_state	text	<p>Synchronization state:</p> <ul style="list-style-type: none"> <li>• <b>async</b>: asynchronous replication.</li> <li>• <b>sync</b>: synchronous replication.</li> <li>• <b>potential</b>: The standby node is asynchronous currently, but if a current synchronization node fails, the standby node becomes synchronous.</li> <li>• <b>quorum</b>: switches between the synchronous and asynchronous states to ensure that there are more than a certain number of synchronous standby nodes. Generally, the number of synchronous standby nodes is <math>(n+1)/2-1</math>, where <math>n</math> indicates the total number of copies. Whether the standby node is synchronous depends on whether logs are received first. For details, see the description of the <b>synchronous_standby_names</b> parameter.</li> </ul>

### 12.3.209 PG\_STAT\_SYS\_INDEXES

PG\_STAT\_SYS\_INDEXES displays index status information about all the system catalogs in pg\_catalog and information\_schema.

**Table 12-311** PG\_STAT\_SYS\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index.
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema that the index is in.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans initiated on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 12.3.210 PG\_STAT\_SYS\_TABLES

PG\_STAT\_SYS\_TABLES displays statistics about the system catalogs of all the namespaces in pg\_catalog and information\_schema.

**Table 12-312** PG\_STAT\_SYS\_TABLES columns

Name	Type	Description
relid	oid	OID of a table.
schemaname	name	Name of the schema that the table is in.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.

Name	Type	Description
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time when this table was manually vacuumed (excluding VACUUM FULL).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting VACUUM FULL).
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times the table has been manually analyzed.
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon.
last_data_changed	timestamp with time zone	Last modification time of the table data.

### 12.3.211 PG\_STAT\_USER\_FUNCTIONS

**PG\_STAT\_USER\_FUNCTIONS** shows user-defined function status information in the namespace. (The language of the function is non-internal language.)

**Table 12-313** PG\_STAT\_USER\_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function

Name	Type	Description
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times the function has been called
total_time	double precision	Total time spent in the function and all other functions called by it
self_time	double precision	Total time spent in the function itself, excluding other functions called by it

### 12.3.212 PG\_STAT\_USER\_INDEXES

PG\_STAT\_USER\_INDEXES displays information about the index status of user-defined ordinary tables and TOAST tables.

**Table 12-314** PG\_STAT\_USER\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index.
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema that the index is in.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans initiated on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 12.3.213 PG\_STAT\_USER\_TABLES

PG\_STAT\_USER\_TABLES displays status information about user-defined ordinary tables and TOAST tables in the namespaces.

**Table 12-315** PG\_STAT\_USER\_TABLES columns

Name	Type	Description
relid	oid	OID of a table.
schemaname	name	Name of the schema that the table is in.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time when this table was manually vacuumed (excluding <b>VACUUM FULL</b> ).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting <b>VACUUM FULL</b> ).

Name	Type	Description
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times the table has been manually analyzed.
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon.
last_data_change_d	timestamp with time zone	Last modification time of the table data.

### 12.3.214 PG\_STAT\_XACT\_ALL\_TABLES

**PG\_STAT\_XACT\_ALL\_TABLES** displays transaction status information about all ordinary tables and TOAST tables in the namespaces.

**Table 12-316** PG\_STAT\_XACT\_ALL\_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

### 12.3.215 PG\_STAT\_XACT\_SYS\_TABLES

**PG\_STAT\_XACT\_SYS\_TABLES** displays transaction status information of the system catalog in the namespace.

**Table 12-317** PG\_STAT\_XACT\_SYS\_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

## 12.3.216 PG\_STAT\_XACT\_USER\_FUNCTIONS

**PG\_STAT\_XACT\_USER\_FUNCTIONS** contains statistics on the execution of each function.

**Table 12-318** PG\_STAT\_XACT\_USER\_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times that the function has been called
total_time	double precision	Total time spent in the function and all other functions called by it
self_time	double precision	Total time spent in the function itself, excluding other functions called by it

## 12.3.217 PG\_STAT\_XACT\_USER\_TABLES

**PG\_STAT\_XACT\_USER\_TABLES** displays transaction status information of the user table in the namespace.

**Table 12-319** PG\_STAT\_XACT\_USER\_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

## 12.3.218 PG\_STATS

**PG\_STATS** displays single-column statistics stored in the `pg_statistic` table. The GUC parameter **autovacuum\_naptime** specifies the interval for updating statistics recorded in the view.

**Table 12-320** PG\_STATS columns

Name	Type	Reference	Description
schemaname	name	<b>nspname</b> in <b>PG_NAMESPACE</b>	Name of the schema that contains a table.
tablename	name	<b>PG_CLASS</b> .relname	Table name.
attname	name	<b>PG_ATTRIBUTE</b> .attname	Field name.
inherited	Boolean	-	Table inheritance is not supported currently. The value of this column is <b>false</b> .

Name	Type	Reference	Description
null_frac	real	-	Percentage of column entries that are null.
avg_width	integer	-	Average width in bytes of column's entries.
n_distinct	real	-	<ul style="list-style-type: none"> <li>Estimated number of distinct values in the column if the value is greater than 0.</li> <li>Negative number obtained by multiplying the result calculated by dividing the number of distinct values by the number of rows by -1 if the value is less than 0. For example, -1 indicates that the number of distinct values is the same as the number of rows for a column combination.               <ol style="list-style-type: none"> <li>The negated form is used when <b>ANALYZE</b> believes that the number of distinct values is likely to increase as the table grows.</li> <li>The positive form is used when the column seems to have a fixed number of possible values.</li> </ol> </li> <li>The number of distinct values is unknown if the value is 0.</li> </ul>
n_dndistinct	real	-	<p>Number of not-null distinct values in the <b>dn1</b> column.</p> <ul style="list-style-type: none"> <li>Exact number of distinct values if the value is greater than 0.</li> <li>Negative number obtained by multiplying the result calculated by dividing the number of distinct values by the number of rows by -1 if the value is less than 0. For example, if the value of a column appears twice in average, set <b>n_dndistinct</b> to -0.5.</li> <li>The number of distinct values is unknown if the value is 0.</li> </ul>

Name	Type	Reference	Description
most_common_vals	anyarray	-	List of the most common values in a column. If this column does not have the most common value, the value is <b>NULL</b> .
most_common_freqs	real[]	-	List of the frequencies of the most common values in a column. The frequencies are obtained by dividing the number of occurrences of each value by the number of rows. ( <b>NULL</b> if <b>most_common_vals</b> is <b>NULL</b> )
histogram_bounds	anyarray	-	Frequency histogram consisting of values excluding null values and MCVs. If a value appears in the value of <b>most_common_vals</b> , it does not appear in the histogram. If the column data type does not have the < operator or the list specified by <b>most_common_vals</b> contains all values of the column, the histogram information of the column is <b>NULL</b> .
correlation	real	-	Correlation between the physical row sequence and logical row sequence of a column value. The value ranges from -1 to +1. When the value is close to -1 or +1, the index scan overhead is less than that when the value is close to 0 because random access to the disk is reduced. This column is <b>NULL</b> if the column data type does not have a < operator.
most_common_elems	anyarray	-	A list of non-null element values most often appearing.
most_common_elem_freqs	real[]	-	List that records the frequency of the most commonly used non-null elements.
elem_count_histogram	real[]	-	A histogram of the counts of distinct non-null element values.

### 12.3.219 PG\_STATIO\_ALL\_INDEXES

PG\_STATIO\_ALL\_INDEXES is used to query information about one row for each index in the current database, showing statistics about I/O on that specific index.

**Table 12-321** PG\_STATIO\_ALL\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index.
indexrelid	oid	OID of this index.
schemaname	name	Name of the schema that this index is in.
relname	name	Name of the table that the index is created for.
indexrelname	name	Index name.
idx_blks_read	bigint	Number of disk blocks read from this index.
idx_blks_hit	bigint	Number of cache hits in this index.

### 12.3.220 PG\_STATIO\_ALL\_SEQUENCES

PG\_STATIO\_ALL\_SEQUENCES displays the I/O statistics of each sequence in the current database.

**Table 12-322** PG\_STATIO\_ALL\_SEQUENCES columns

Name	Type	Description
relid	oid	OID of a sequence.
schemaname	name	Name of the schema that this sequence is in.
relname	name	Name of this sequence.
blks_read	bigint	Number of disk blocks read from this sequence.
blks_hit	bigint	Number of cache hits in this sequence.

### 12.3.221 PG\_STATIO\_ALL\_TABLES

PG\_STATIO\_ALL\_TABLES is used to query I/O statistics of each table (including TOAST tables) in the current database.

**Table 12-323** PG\_STATIO\_ALL\_TABLES columns

Name	Type	Description
relid	oid	OID of a table.
schemaname	name	Name of the schema that this table is in.
relname	name	Table name.
heap_blks_read	bigint	Number of disk blocks read from this table.
heap_blks_hit	bigint	Number of cache hits in this table.
idx_blks_read	bigint	Number of disk blocks read from all indexes in this table.
idx_blks_hit	bigint	Number of cache hits of all indexes in this table.
toast_blks_read	bigint	Number of disk blocks (if any) read from the TOAST table of this table.
toast_blks_hit	bigint	Number of cache hits (if any) in the TOAST table of this table.
tidx_blks_read	bigint	Number of disk blocks (if any) read from the TOAST table indexes of this table.
tidx_blks_hit	bigint	Number of cache hits (if any) in the TOAST table indexes of this table.

### 12.3.222 PG\_STATIO\_SYS\_INDEXES

PG\_STATIO\_SYS\_INDEXES displays I/O status information about all system catalog indexes in the namespaces.

**Table 12-324** PG\_STATIO\_SYS\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for an index.
indexrelid	oid	OID of this index.
schemaname	name	Name of the schema that this index is in.
relname	name	Name of the table that the index is created for.
indexrelname	name	Index name.
idx_blks_read	bigint	Number of disk blocks read from this index.
idx_blks_hit	bigint	Number of cache hits in this index.

### 12.3.223 PG\_STATIO\_SYS\_SEQUENCES

PG\_STATIO\_SYS\_SEQUENCES displays I/O status information about all the sequences in the namespaces.

**Table 12-325** PG\_STATIO\_SYS\_SEQUENCES columns

Name	Type	Description
relid	oid	OID of a sequence.
schemaname	name	Name of the schema that this sequence is in.
relname	name	Name of this sequence.
blks_read	bigint	Number of disk blocks read from this sequence.
blks_hit	bigint	Number of cache hits in this sequence.

### 12.3.224 PG\_STATIO\_SYS\_TABLES

PG\_STATIO\_SYS\_TABLES shows I/O status information about all the system catalogs in the namespaces.

**Table 12-326** PG\_STATIO\_SYS\_TABLES columns

Name	Type	Description
relid	oid	OID of a table.
schemaname	name	Name of the schema that this table is in.
relname	name	Table name.
heap_blks_read	bigint	Number of disk blocks read from this table.
heap_blks_hit	bigint	Number of cache hits in this table.
idx_blks_read	bigint	Number of disk blocks read from all indexes in this table.
idx_blks_hit	bigint	Number of cache hits of all indexes in this table.
toast_blks_read	bigint	Number of disk blocks (if any) read from the TOAST table of this table.
toast_blks_hit	bigint	Number of cache hits (if any) in the TOAST table of this table.

Name	Type	Description
tidx_blks_read	bigint	Number of disk blocks (if any) read from the TOAST table indexes of this table.
tidx_blks_hit	bigint	Number of cache hits (if any) in the TOAST table indexes of this table.

### 12.3.225 PG\_STATIO\_USER\_INDEXES

PG\_STATIO\_USER\_INDEXES displays I/O status information about all the user relationship table indexes in the namespaces.

**Table 12-327** PG\_STATIO\_USER\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index.
indexrelid	oid	OID of this index.
schemaname	name	Name of the schema that this index is in.
relname	name	Name of the table for this index.
indexrelname	name	Index name.
idx_blks_read	bigint	Number of disk blocks read from this index.
idx_blks_hit	bigint	Number of cache hits in this index.

### 12.3.226 PG\_STATIO\_USER\_SEQUENCES

PG\_STATIO\_USER\_SEQUENCES shows I/O status information about all the user relationship table sequences in the namespaces.

**Table 12-328** PG\_STATIO\_USER\_SEQUENCES columns

Name	Type	Description
relid	oid	OID of a sequence.
schemaname	name	Name of the schema that this sequence is in.
relname	name	Name of this sequence.

Name	Type	Description
blks_read	bigint	Number of disk blocks read from this sequence.
blks_hit	bigint	Number of cache hits in this sequence.

### 12.3.227 PG\_STATIO\_USER\_TABLES

PG\_STATIO\_USER\_TABLES displays I/O status information about all the user relationship tables in the namespaces.

**Table 12-329** PG\_STATIO\_USER\_TABLES columns

Name	Type	Description
relid	oid	OID of a table.
schemaname	name	Name of the schema that this table is in.
relname	name	Table name.
heap_blks_read	bigint	Number of disk blocks read from this table.
heap_blks_hit	bigint	Number of cache hits in this table.
idx_blks_read	bigint	Number of disk blocks read from all indexes in this table.
idx_blks_hit	bigint	Number of cache hits of all indexes in this table.
toast_blks_read	bigint	Number of disk blocks (if any) read from the TOAST table of this table.
toast_blks_hit	bigint	Number of cache hits (if any) in the TOAST table of this table.
tidx_blks_read	bigint	Number of disk blocks (if any) read from the TOAST table indexes of this table.
tidx_blks_hit	bigint	Number of cache hits (if any) in the TOAST table indexes of this table.

### 12.3.228 PG\_THREAD\_WAIT\_STATUS

PG\_THREAD\_WAIT\_STATUS allows you to test the blocking status about the backend thread and auxiliary thread of the current instance.

**Table 12-330** PG\_THREAD\_WAIT\_STATUS columns

Name	Type	Description
node_name	text	Current node name.
db_name	text	Database name.
thread_name	text	Thread name.
query_id	bigint	Query ID. It is equivalent to <b>debug_query_id</b> .
tid	bigint	Thread ID of the current thread.
sessionid	bigint	Current session ID.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent session ID.
tlevel	integer	Level of the streaming thread.
smpid	integer	Concurrent thread ID.
wait_status	text	Waiting status of the current thread. For details about the waiting status, see <a href="#">Table 12-331</a> .
wait_event	text	If <b>wait_status</b> is set to <b>acquire lock</b> , <b>acquire lwlock</b> , or <b>wait io</b> , this column describes the lock, lightweight lock, and I/O information, respectively. Otherwise, this column is empty.
locktag	text	Information about the lock that the current thread is waiting to obtain.
lockmode	text	Lock mode that the current thread is waiting to obtain. The values include modes in the table-level lock, row-level lock, and page-level lock.
block_session_id	bigint	ID of the session that blocks the current thread from obtaining the lock.
global_session_id	text	Global session ID.

The waiting states in the **wait\_status** column are as follows:

**Table 12-331** Waiting state list

Value	Description
none	Waiting for no event.
acquire lock	Waiting for locking until the locking succeeds or times out.

Value	Description
acquire lwlock	Waiting for a lightweight lock.
wait io	Waiting for I/O completion.
wait cmd	Waiting for reading network communication packets to complete.
wait pooler get conn	Waiting for pooler to obtain connections.
wait pooler abort conn	Waiting for pooler to terminate connections.
wait pooler clean conn	Waiting for pooler to clear connections.
pooler create conn: [nodename], total N	Waiting for the pooler to set up a connection. The connection is being established with the node specified by <i>nodename</i> , and there are <i>N</i> connections waiting to be set up.
get conn	Obtaining the connection to other nodes.
set cmd: [nodename]	Waiting for running the <b>SET</b> , <b>RESET</b> , <b>TRANSACTION BLOCK LEVEL PARA SET</b> , or <b>SESSION LEVEL PARA SET</b> statement on the connection. The statement is being executed on the node specified by <i>nodename</i> .
cancel query	Canceling the SQL statement that is being executed through the connection.
stop query	Stopping the query that is being executed through the connection.
wait node: [nodename](plevel), total N, [phase]	Waiting for receiving data from a connected node. The thread is waiting for data from the <i>plevel</i> thread of the node specified by <i>nodename</i> . The data of <i>N</i> connections is waiting to be returned. If <i>phase</i> is included, the possible phases are as follows: <ul style="list-style-type: none"> <li>● <b>begin</b>: The transaction is being started.</li> <li>● <b>commit</b>: The transaction is being committed.</li> <li>● <b>rollback</b>: The transaction is being rolled back.</li> </ul>
wait transaction sync: xid	Waiting for synchronizing the transaction specified by <i>xid</i> .
wait wal sync	Waiting for the completion of WAL of synchronization from the specified LSN to the standby instance.

Value	Description
wait data sync	Waiting for the completion of data page synchronization to the standby instance.
wait data sync queue	Waiting for putting the data pages that are in the row-store into the synchronization queue.
flush data: [nodename](plevel), [phase]	Waiting for sending data to the plevel thread of the node specified by <i>nodename</i> . If <i>phase</i> is included, the possible phase is <b>wait quota</b> , indicating that the current communication flow is waiting for the quota value.  A quota indicates the size of data that can be received by the network channel. <b>wait quota</b> indicates that the sender (data producer) waits for the receiver (data consumer) to send the quota information of the current connection.
stream get conn: [nodename], total N	Waiting for connecting to the consumer object of the node specified by <i>nodename</i> when the stream flow is initialized. There are <i>N</i> consumers waiting to be connected.
wait producer ready: [nodename] (plevel), total N	Waiting for each producer to be ready when the stream flow is initialized. The thread is waiting for the procedure of the plevel thread on the <i>nodename</i> node to be ready. There are <i>N</i> producers waiting to be ready.
synchronize quit	Waits for the threads in the stream thread group to exit when the stream plan ends.
nodegroup destroy	Waiting for destroying the stream node group when the stream plan ends.
wait active statement	Waiting for job execution under resource and load control.
gtm connect	Waiting for connecting to GTM.
gtm get gxid	Waiting for obtaining XIDs from GTM.
gtm get snapshot	Waiting for obtaining transaction snapshots from GTM.
gtm begin trans	Waiting for GTM to start a transaction.
gtm commit trans	Waiting for GTM to commit a transaction.
gtm rollback trans	Waiting for GTM to roll back a transaction.

Value	Description
gtm start prepare trans	Waiting for GTM to start the prepare phase of a two-phase transaction.
gtm prepare trans	Waiting for GTM to complete the prepare phase of a two-phase transaction.
gtm open sequence	Waiting for GTM to open a sequence.
gtm close sequence	Waiting for GTM to close a sequence.
gtm create sequence	Waiting for GTM to create a sequence.
gtm alter sequence	Waiting for GTM to modify a sequence.
gtm get sequence val	Waiting for obtaining the next value of a sequence from GTM.
gtm set sequence val	Waiting for GTM to set a sequence value.
gtm drop sequence	Waiting for GTM to delete a sequence.
gtm rename sequence	Waiting for GTM to rename a sequence.
analyze: [relname], [phase]	Executing ANALYZE for the <i>relname</i> table. If <i>phase</i> is included, the possible phase is <b>autovacuum</b> , indicating that the database automatically enables the AutoVacuum thread to execute <b>ANALYZE</b> .
vacuum: [relname], [phase]	Executing VACUUM for the <i>relname</i> table. If <i>phase</i> is included, the possible phase is <b>autovacuum</b> , indicating that the database automatically enables the AutoVacuum thread to execute <b>VACUUM</b> .
vacuum full: [relname]	Executing VACUUM FULL for the <i>relname</i> table.
create index	An index is being created.
HashJoin - [ build hash   write file ]	The <b>HashJoin</b> operator is being executed. In this phase, you need to pay attention to the execution time-consuming. <ul style="list-style-type: none"><li>• <b>build hash</b>: The <b>HashJoin</b> operator is creating a hash table.</li><li>• <b>write file</b>: The <b>HashJoin</b> operator is writing data to disks.</li></ul>

Value	Description
HashAgg - [ build hash   write file ]	The <b>HashAgg</b> operator is being executed. In this phase, you need to pay attention to the execution time-consuming. <ul style="list-style-type: none"> <li>● <b>build hash</b>: The <b>HashAgg</b> operator is creating a hash table.</li> <li>● <b>write file</b>: The <b>HashAgg</b> operator is writing data to disks.</li> </ul>
HashSetop - [build hash   write file ]	The <b>HashSetop</b> operator is being executed. In this phase, you need to pay attention to the execution time-consuming. <ul style="list-style-type: none"> <li>● <b>build hash</b>: The <b>HashSetop</b> operator is creating a hash table.</li> <li>● <b>write file</b>: The <b>HashSetop</b> operator is writing data to disks.</li> </ul>
Sort   Sort - [fetch tuple   write file]	The <b>Sort</b> operator is used for sorting. <b>fetch tuple</b> indicates that the <b>Sort</b> operator is obtaining tuples, and <b>write file</b> indicates that the <b>Sort</b> operator is writing data to disks.
Material   Material - write file	The <b>Material</b> operator is being executed. <b>write file</b> indicates that the <b>Material</b> operator is writing data to disks.
standby read recovery conflict	The read-only mode of the standby node conflicts with the log replay mode.
standby get snapshot	The standby node obtains the snapshot in read-only mode.
prune table	Waiting for a heap table to clear historical deleted data.
prune index	Waiting for an index to clear historical deleted data.
vacuum gpi	Waiting for a GPI to clear historical deleted data.
gtm reset xmin	Waiting for GTM to reset the minimum transaction ID.
gtm get xmin	Waiting for obtaining the minimum transaction ID from GTM.
gtm get csn	Waiting for obtaining the CSN from GTM.
gtm start prepare trans	Waiting for the GTM to start a two-phase transaction.
gtm rename sequence	Waiting for GTM to rename a sequence.

Value	Description
wait reserve td	Waiting for an Ustore transaction slot allocation.
wait td rollback	Waiting for an Ustore transaction slot to roll back transactions.
wait available td	Waiting for an available transaction slot for Ustore.
wait transaction rollback	Waiting for a transaction to roll back.
gtm set disaster cluster	Waiting for setting the DR cluster information on the GTM.
gtm get disaster cluster	Waiting for obtaining the DR cluster information from the GTM.
gtm del disaster cluster	Waiting for deleting the DR cluster information from the GTM.
gtm set consistency point	Waiting for setting the barrier in GTM.
wait sync bgworkers	Waiting for an index worker thread created in parallel to complete local scanning and sorting.
security audit write pipe	Waiting for an audit log to be written to the pipe.
wait unpack stream plan	Waiting for parsing stream plan on the DN.
wait fetch undo record	Waiting for reading the target undo record.
wait heap hot search buffer	Waiting for reading the Astore tuple that meets the snapshot requirements through the hot link.
wait exclusive lwlock	The starvation prevention mechanism is triggered. A new lightweight lock request waits for the previously blocked lightweight lock to be obtained.

If **wait\_status** is **acquire lwlock**, **acquire lock**, or **wait io**, there is an event performing I/O operations or waiting for obtaining the corresponding lightweight lock or transaction lock.

The following table describes the corresponding wait events when **wait\_status** is **acquire lwlock**. If **wait\_event** is **extension**, the lightweight lock is dynamically allocated and is not monitored.

**Table 12-332** List of wait events corresponding to lightweight locks

wait_event	Description
ShmemIndexLock	Used to protect the primary index table, a hash table, in shared memory.
OidGenLock	Used to prevent different threads from generating the same OID.
XidGenLock	Used to prevent two transactions from obtaining the same XID.
ProcArrayLock	Used to prevent concurrent access to or concurrent modification on the ProcArray shared array.
SInvalReadLock	Used to prevent concurrent execution with invalid message deletion.
SInvalWriteLock	Used to prevent concurrent execution with invalid message write and deletion.
WALInsertLock	Used to prevent concurrent execution with WAL insertion.
WALWriteLock	Used to prevent concurrent write from a WAL buffer to a disk.
ControlFileLock	Used to prevent concurrent read/write or concurrent write/write on the <b>pg_control</b> file.
CheckpointLock	Used to prevent multi-checkpoint concurrent execution.
CLogControlLock	Used to prevent concurrent access to or concurrent modification on the Clog control data structure.
SubtransControlLock	Used to prevent concurrent access to or concurrent modification on the subtransaction control data structure.
MultiXactGenLock	Used to allocate a unique MultiXact ID in serial mode.
MultiXactOffsetControlLock	Used to prevent concurrent read/write or concurrent write/write on <b>pg_multixact/offset</b> .
MultiXactMemberControlLock	Used to prevent concurrent read/write or concurrent write/write on <b>pg_multixact/members</b> .
RelCacheInitLock	Used to add a lock before any operations are performed on the <b>init</b> file when messages are invalid.

<b>wait_event</b>	<b>Description</b>
CheckpointCommLock	Used to send file flush requests to a checkpoint. The request structure needs to be inserted to a request queue in serial mode.
TwoPhaseStateLock	Used to prevent concurrent access to or modification on two-phase information sharing arrays.
TablespaceCreateLock	Used to check whether a tablespace already exists.
BtreeVacuumLock	Used to prevent VACUUM from clearing pages that are being used by B-tree indexes.
AlterPortLock	Used to protect the CN's operation of changing the registered port number.
AutovacuumLock	Used to access the autovacuum worker array in serial mode.
AutovacuumScheduleLock	Used to distribute tables requiring VACUUM in serial mode.
AutoanalyzeLock	Used to obtain and release resources related to a task that allows for autoanalyze execution.
SyncScanLock	Used to determine the start position of a relfilenode during heap scanning.
NodeTableLock	Used to protect a shared structure that stores CN and DN node information.
PoolerLock	Used to prevent two threads from simultaneously obtaining the same connection from a connection pool.
RelationMappingLock	Used to wait for the mapping file between system catalogs and storage locations to be updated.
Async Ctl	Used to protect asynchronization buffers.
AsyncCtlLock	Used to prevent concurrent access to or concurrent modification on the sharing notification status.
AsyncQueueLock	Used to prevent concurrent access to or concurrent modification on the sharing notification queue.
SerializableXactHashLock	Used to prevent concurrent read/write or concurrent write/write on a sharing structure for serializable transactions.

<b>wait_event</b>	<b>Description</b>
SerializableFinishedListLock	Used to prevent concurrent read/write or concurrent write/write on a shared linked list for completed serial transactions.
SerializablePredicateLockList-Lock	Used to protect a linked list of serializable transactions that have locks.
OldSerXidLock	Used to protect a structure that records serializable transactions that have conflicts.
FileStatLock	Used to protect a data structure that stores statistics file information.
SyncRepLock	Used to protect Xlog synchronization information during primary/standby replication.
DataSyncRepLock	Used to protect data page synchronization information during primary/standby replication.
MetaCacheSweepLock	Used to add a lock when metadata is cyclically washed out.
ExtensionConnectorLibLock	Used to add a lock when a specific dynamic library is loaded or uninstalled in ODBC connection initialization scenarios.
SearchServerLibLock	Used to add a lock on the file read operation when a specific dynamic library is initially loaded in GPU-accelerated scenarios.
LsnXlogChkFileLock	Used to serially update the Xlog flush points for primary and standby nodes recorded in a specific structure.
GTMHostInfoLock	Used to prevent concurrent access to or concurrent modification on GTM host information.
ReplicationSlotAllocation-Lock	Used to add a lock when a primary node allocates streaming replication slots during primary/standby replication.
ReplicationSlotControlLock	Used to prevent concurrent update of streaming replication slot status during primary/standby replication.
ResourcePoolHashLock	Used to prevent concurrent access to or concurrent modification on a resource pool table, a hash table.
OBSGetPathLock	Used to prevent concurrent read/write or concurrent write/write on an OBS path.

<b>wait_event</b>	<b>Description</b>
JobShmemLock	Used to protect global variables in the shared memory that is periodically read during a scheduled job where MPP is compatible with Oracle Database.
OBSRuntimeLock	Used to obtain environment variables, for example, <i>GAUSSHOME</i> .
CriticalCacheBuildLock	Used to load caches from a shared or local cache initialization file.
WaitCountHashLock	Used to protect a shared structure in user statement counting scenarios.
BufMappingLock	Used to protect operations on a shared-buffer mapping table.
LockMgrLock	Used to protect a common lock structure.
PredicateLockMgrLock	Used to protect a lock structure that has serializable transactions.
OperatorRealTLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the operator level.
OperatorHistLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the operator level.
SessionRealTLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the query level.
SessionHistLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the query level.
CacheSlotMappingLock	Used to protect global CU cache information.
BarrierLock	Used to ensure that only one thread is creating a barrier at a time.
GPCCommitLock	Used to protect the addition of the global plan cache hash table.
GPCClearLock	Used to protect the clearing of the global plan cache hash table.
GPCTimelineLock	Used to protect the timeline check of the global plan cache hash table.
GPCMappingLock	Used to manage the global plan cache.

<b>wait_event</b>	<b>Description</b>
GPCPrepareMappingLock	Used to manage the global plan cache.
GPRCMappingLock	Used to manage the access and modification operations of the global cache hash table of autonomous transactions.
BufFreelistLock	Used to ensure the atomicity of free list operations in the shared buffer.
AddinShmemInitLock	Used to protect the initialization of the shared memory object.
wait active statement	Waiting for job execution under resource and load control.
wait memory	Waiting for obtaining the memory.
DnUsedSpaceHashLock	Used to update space usage information corresponding to a session.
InstanceRealTLock	Used to protect the update of the hash table that stores shared instance statistics.
IOStatLock	Used to concurrently maintain the hash table of resource management I/O statistics.
PldebugLock	Used to debug stored procedures and perform concurrent maintenance operations.
StartBlockMappingLock	Used by globalstat to obtain information such as startblockarray from pgstat.
GlobalSeqLock	Used to manage global sequence numbers.
MatviewSeqnoLock	Used to manage the cache of materialized views.
DataFileIdCacheLock	Used to manage the concurrent access and storage of hash table that stores data file descriptor in the shared memory.
GTMHostInfoLock	Used to protect the concurrent access and storage of shared GTM host information.
TwoPhaseStatePartLock	Used to protect the status information of two-phase transactions (in each partition).
WALBufMappingLock	Used to protect the mapping between WAL buffer page and LSN offset.
UndoZoneLock	Used to protect the concurrent access and storage of undo zone.
RollbackReqHashLock	Used to protect the concurrent access and storage of hash table that stores the rollback request in the shared memory.

<b>wait_event</b>	<b>Description</b>
UHeapStatLock	Used to protect the concurrent access and storage of Ustore statistics.
WALWritePaxosLock	Used to protect the concurrent sequence of WALs written to the paxos replication component.
SyncPaxosLock	Used to protect the concurrent access and storage of paxos synchronization queue.
BackgroundWorkerLock	Used to protect the concurrent sequence of background workers.
HadrSwitchoverLock	Used to protect the concurrent sequence of DR switchover.
HashUidLock	Used to protect the concurrent sequence of UID allocation.
ParallelDecodeLock	Used to protect the concurrent sequence of parallel decoding.
XLogMaxCSNLock	Used to protect the maximum volume of CSNs that can be restored locally in DR mode.
DisasterCacheLock	Used to protect the concurrent access and storage of DR cache information in the shared memory.
MaxCSNArrayLock	Used to protect the restoration progress of standby node CSNs in each shard in the shared memory.
RepairBadBlockStatHashLock	Used to protect the concurrent access and storage of the hash table that stores damaged pages in the shared memory.
DropArchiveSlotLock	Used to protect the concurrent sequence of deleting the archive slot.
ProcXactMappingLock	Used to protect the concurrent access and storage of the hash table that stores the mapping information between transaction IDs and threads.
UndoPerZoneLock	Used to protect the concurrent access and storage of each information in undo zone.
UndoSpaceLock	Used to protect the concurrent access and storage of undo space.
SnapshotBlockLock	Used to control the concurrent sequence of snapshot-based backup and disk flushing.

<b>wait_event</b>	<b>Description</b>
DWSingleFlushFirstLock	Used to control the concurrent sequence of segment-pages and single-page doublewrite files.
RestartPointQueueLock	Used to control the concurrent access and storage of restart point arrays on the standby node.
UnlinkRelHashTblLock	Used to protect the concurrent access and storage of the hash table that restores files to be deleted.
UnlinkRelForkHashTblLock	Used to protect the concurrent access and storage of the hash table that restores the fork files to be deleted.
WALFlushWait	Used to protect the concurrent sequence of log flushing.
WALConsensusWait	Used to protect the transaction commit or log replay which is performed only when the logs are consistent.
WALBufferInitWait	Used to protect the initialization and disk flushing sequence of WAL pages in the shared memory.
WALInitSegment	Used to protect the initialization sequence of WAL segment files.
PgwrSyncQueueLock	Used to protect the concurrent access and storage of file queues to be flushed to disks.
BarrierHashTblLock	Used to protect the concurrent access and storage of barrier tables in the shared memory.
PageRepairHashTblLock	Used to protect the concurrent access and storage of the hash table that stores repaired pages.
FileRepairHashTblLock	Used to protect the concurrent access and storage of the hash table that stores repaired files.
BadBlockStatHashLock	Used to protect the concurrent access and storage of the hash table that stores damaged pages in the shared memory.
BufferIOLock	Used to protect the concurrent I/O for loading or evicting a single page in the shared buffer.
BufferContentLock	Used to protect the concurrent read and write operations on a single page in the shared buffer.

<b>wait_event</b>	<b>Description</b>
CUSlotListLock	Used to protect the concurrent access and storage of the slot linked list in the shared CU.
DataCacheLock	Used to protect the concurrent access and storage of the read-only cache in the shared CU.
MetaCacheLock	Used to protect the concurrent access and storage of the meta cache in the shared CU.
CBMParseXlogLock	Used to control the concurrent access and storage of CBM files.
CLogBufMappingLock	Used to control the concurrent access and storage of mapping in the shared clog page.
CLOG Ctl	Used to control the concurrent access and storage of each clog partition.
CSNBufMappingLock	Used to control the concurrent access and storage of mapping in the shared csnolog page.
CSNLOG Ctl	Used to control the concurrent access and storage of each csnolog partition.
DelayDDLLock	Used to control the concurrent sequence of DDL and delay backup function of the deletion.
DoubleWriteLock	Used to control the concurrent sequence of double write module.
DfsConnectorCacheLock	Used to control the concurrent access and storage of dfs connection cache.
DfsUserLoginLock	Used to control the concurrent sequence of dfs user login.
DfsSpaceCacheLock	Used to control the concurrent sequence of dfs space management cache.
PGPROCLock	Used to protect the concurrent access and storage of global shared thread array.
RelfilenodeReuseLock	Used to control the concurrent sequence of file name reuse.
ReplicationSlotLock	Used to protect the concurrent access and storage of logical replication slot.
LogicalReplicationSlotPersistentDataLock	Used to control the concurrent access and storage of logical replication slot persistence.
RowPageReplicationLock	Used to control the concurrent sequence of row-store page replication.
MultiXactOffset Ctl	Used to protect the concurrent access and storage of multixact offset document.

<b>wait_event</b>	<b>Description</b>
MultiXactMember Ctl	Used to protect the concurrent access and storage of multixact member document.
OldSerXid SLRU Ctl	Used to protect the concurrent access and storage of oldser transaction ID cache.
FullBuildXlogCopyStartPtr-Lock	Used to control the concurrent access and storage of the start point of full build request.
RcvWriteLock	Used to control the concurrent sequence of WAL receiver writer.
XlogRemoveSegLock	Used to control the concurrent access and storage of latest deleted WAL document.
CsnMinLock	Used to control the concurrent access and storage of CSN MIN
HypoIndexLock	Used to create, delete and reset the virtual index by the lightweight lock.
XGBoostLibLock	Used by DB4AI to call the XGBoost library.
InstrUserLockId	Used to protect the concurrent modification of the hash table that stores user login and logout records.
GsStackLock	Used to ensure that the gs_stack function is not concurrently called.
InstrStmtTrackCtlLock	Used to protect the concurrent access and storage of the hash table when full SQL statements are dynamically enabled.
CaptureViewFileHashLock	Used to protect the concurrent access and storage of the hash table when capturing the performance view.
UniqueSqlEvictLock	Used to protect the concurrent access and storage of the hash table when the unique SQL reclamation is enabled.
ASPMappingLock	Used to manage the concurrent access and storage of the ASP hash table.
AuditIndexFileLock	Used to protect concurrent read and write of index files in the audit log.
SQLAdvisorLock	Used to manage the concurrent access and storage of recommended hash tables in distribution key management.
LWTRANCHE_ACCOUNT_TABLE	Used to control the concurrent read and write of the hash table that stores the account locking status.

The following table describes the corresponding wait events when **wait\_status** is **wait io**.

**Table 12-333** List of wait events corresponding to I/Os

<b>wait_event</b>	<b>Description</b>
BufFileRead	Reads data from a temporary file to a specified buffer.
BufFileWrite	Writes the content of a specified buffer to a temporary file.
ControlFileRead	Reads the <b>pg_control</b> file, mainly during database startup, checkpoint execution, and primary/standby verification.
ControlFileSync	Flushes the <b>pg_control</b> file to a disk, mainly during database initialization.
ControlFileSyncUpdate	Flushes the <b>pg_control</b> file to a disk, mainly during database startup, checkpoint execution, and primary/standby verification.
ControlFileWrite	Writes the <b>pg_control</b> file, mainly during database initialization.
ControlFileWriteUpdate	Updates the <b>pg_control</b> file, mainly during database startup, checkpoint execution, and primary/standby verification.
CopyFileRead	Reads a file during file copying.
CopyFileWrite	Writes a file during file copying.
DataFileExtend	Writes a file during file name extension.
DataFileFlush	Flushes a table data file to a disk.
DataFileImmediateSync	Flushes a table data file to a disk immediately.
DataFilePrefetch	Reads a table data file asynchronously.
DataFileRead	Reads a table data file synchronously.
DataFileSync	Flushes table data file modifications to a disk.
DataFileTruncate	Truncates a table data file.
DataFileWrite	Writes a table data file.
LockFileAddToDataDirRead	Reads the <b>postmaster.pid</b> file.
LockFileAddToDataDirSync	Flushes the <b>postmaster.pid</b> file to a disk.

<b>wait_event</b>	<b>Description</b>
LockFileAddToDataDirWrite	Writes PID information into the <b>postmaster.pid</b> file.
LockFileCreateRead	Read the LockFile file <b>%s.lock</b> .
LockFileCreateSync	Flushes the LockFile file <b>%s.lock</b> to a disk.
LockFileCreateWRITE	Writes PID information into the LockFile file <b>%s.lock</b> .
NgroupDestoryLock	Adds a lock to the concurrent modification of the Node Group hash table.
NGroupMappingLock	Adds a lock to the concurrent modification of a single bucket that protects the NodeGroup hash table.
RelationMapRead	Reads the mapping file between system catalogs and storage locations.
RelationMapSync	Flushes the mapping file between system catalogs and storage locations to a disk.
RelationMapWrite	Writes the mapping file between system catalogs and storage locations.
ReplicationSlotRead	Reads a streaming replication slot file during a restart.
ReplicationSlotRestoreSync	Flushes a streaming replication slot file to a disk during a restart.
ReplicationSlotSync	Flushes a temporary streaming replication slot file to a disk during checkpoint execution.
ReplicationSlotWrite	Writes a temporary streaming replication slot file during checkpoint execution.
SLRUFlushSync	Flushes the <b>pg_clog</b> , <b>pg_subtrans</b> , and <b>pg_multixact</b> files to a disk, mainly during checkpoint execution and database shutdown.
SLRURead	Reads the <b>pg_clog</b> , <b>pg_subtrans</b> , and <b>pg_multixact</b> files.
SLRUSync	Writes dirty pages into the <b>pg_clog</b> , <b>pg_subtrans</b> , and <b>pg_multixact</b> files, and flushes the files to a disk, mainly during checkpoint execution and database shutdown.
SLRUWrite	Writes the <b>pg_clog</b> , <b>pg_subtrans</b> , and <b>pg_multixact</b> files.
TimelineHistoryRead	Reads the timeline history file during database startup.

<b>wait_event</b>	<b>Description</b>
TimelineHistorySync	Flushes the <b>timelinehistory</b> file to a disk during database startup.
TimelineHistoryWrite	Writes to the timeline history file during database startup.
TwophaseFileRead	Reads the <b>pg_twophase</b> file, mainly during two-phase transaction commit and restoration.
TwophaseFileSync	Flushes the <b>pg_twophase</b> file to a disk, mainly during two-phase transaction commit and restoration.
TwophaseFileWrite	Writes the <b>pg_twophase</b> file, mainly during two-phase transaction commit and restoration.
WALBootstrapSync	Flushes an initialized WAL file to a disk during database initialization.
WALBootstrapWrite	Writes an initialized WAL file during database initialization.
WALCopyRead	Reads operation generated when an existing WAL file is read for replication after archiving and restoration.
WALCopySync	Flushes a replicated WAL file to a disk after archiving and restoration.
WALCopyWrite	Writes operation generated when an existing WAL file is read for replication after archiving and restoration.
WALInitSync	Flushes a newly initialized WAL file to a disk during log reclaiming or writing.
WALInitWrite	Initializes a newly created WAL file to <b>0</b> during log reclaiming or writing.
WALRead	Reads data from Xlogs during redo operations on two-phase files.
WALSyncMethodAssign	Flushes all open WAL files to a disk.
WALWrite	Writes a WAL file.
DoubleWriteFileRead	Waits for reading the doublewrite file.
DoubleWriteFileSync	Waits for flushing the doublewrite file to the disk.
DoubleWriteFileWrite	Waits for writing the doublewrite file.
PredoProcessPending	Waits for the replay of the remaining records processing in parallel.

<b>wait_event</b>	<b>Description</b>
PredoApply	Waits for the replay of the application in parallel.
DisableConnectFileRead	Waits for reading the lock segment file.
DisableConnectFileSync	Waits for flushing lock segment files to the disk.
DisableConnectFileWrite	Waits for writing the lock segment file
BufHashTableSearch	Used to search the hash table in the shared buffer (page eviction may be triggered).
buffer_strategy_get	Used to obtain pages from strategy buffer (page eviction may be triggered).
UndoFileExtend	Used to extend an undo file.
UndoFilePrefetch	Used to prefetch an undo file.
UndoFileRead	Used to read an undo file.
UndoFileWrite	Used to write an undo file.
UndoFileSync	Flushes an undo file to a disk.
UndoFileUnlink	Deletes an undo file.
UndoMetaSync	Flushes an undo metadata file to a disk.
WALBufferAccess	Accesses WAL buffer. (To ensure performance, only the number of access times is counted in the kernel code, and the access duration is not counted.)
WALBufferFull	Writes WAL files when the WAL buffer is full.
DWSingleFlushGetPos	Searches for an available location for a single-page doublewrite file.
DWSingleFlushWrite	Flushes a single-page doublewrite file to a disk.
MPFL_INIT	Initiates max_page_flush_lsn.
MPFL_READ	Reads max_page_flush_lsn.
MPFL_WRITE	Writes max_page_flush_lsn.
OBSList	Traverses an OBS directory.
OBSRead	Reads an OBS object.
OBSWrite	Writes an OBS object.
LOGCTRL_SLEEP	Waits for the standby node to catch up with logs.
ShareStorageWalRead	Reads log files from the shared disk.

wait_event	Description
ShareStorageWalWrite	Writes log files to the shared disk.
ShareStorageCtlInfoRead	Reads control information from the shared disk.
ShareStorageCtlInfoWrite	Writes control information to the shared disk.

The following table describes the corresponding wait events when **wait\_status** is **acquire lock**.

**Table 12-334** List of wait events corresponding to transaction locks

wait_event	Description
relation	Adds a lock to a table.
extend	Adds a lock to a table being scaled out.
partition	Adds a lock to a partitioned table.
partition_seq	Adds a lock to a partition of a partitioned table.
page	Adds a lock to a table page.
tuple	Adds a lock to a tuple on a page.
transactionid	Adds a lock to a transaction ID.
virtualxid	Adds a lock to a virtual transaction ID.
object	Adds a lock to an object.
userlock	Adds a lock to a user.
advisory	Adds an advisory lock.
filenode	Locks a file name.
subtransactionid	Adds a lock to a sub-transaction ID.
tuple_uid	Used to add a lock to the UID hidden column in the tuple header.

### 12.3.229 PG\_TABLES

PG\_TABLES is used to query useful information about each table in the database.

**Table 12-335** PG\_TABLES columns

Name	Type	Reference	Description
schemaname	name	<b>nspname</b> in <a href="#">PG_NAMESPACE</a>	Name of the schema that contains a table.
tablename	name	<b>relname</b> in <a href="#">PG_CLASS</a>	Table name.
tableowner	name	<b>pg_get_userbyid(relowner)</b> in <a href="#">PG_CLASS</a>	Table owner.
tablespace	name	<b>spcname</b> in <a href="#">PG_TABLESPACE</a>	Tablespace that contains the table (default value: <b>NULL</b> ).
hasindexes	Boolean	<b>relhasindex</b> in <a href="#">PG_CLASS</a>	Specifies whether the table has (or recently had) an index. If it does, the value is <b>TRUE</b> . Otherwise, the value is <b>FALSE</b> .
hasrules	Boolean	<b>relhasruls</b> in <a href="#">PG_CLASS</a>	Specifies whether the table has rules. If it does, the value is <b>TRUE</b> . Otherwise, the value is <b>FALSE</b> .
hastriggers	Boolean	<b>RELHASTRIGGERS</b> in <a href="#">PG_CLASS</a>	Specifies whether the table has triggers. If it does, the value is <b>TRUE</b> . Otherwise, the value is <b>FALSE</b> .
tablecreator	name	<b>pg_get_userbyid(creator)</b> in <a href="#">PG_OBJECT</a>	Table owner.
created	timestamp with time zone	<b>ctime</b> in <a href="#">PG_OBJECT</a>	Time when the table is created.
last_ddl_time	timestamp with time zone	<b>mtime</b> in <a href="#">PG_OBJECT</a>	Time when the last DDL operation is performed on the table.

### 12.3.230 PG\_TDE\_INFO

PG\_TDE\_INFO displays encryption information of the entire cluster.

**Table 12-336** PG\_TDE\_INFO columns

Name	Type	Description
is_encrypt	Boolean	Specifies whether the cluster is an encryption cluster. <ul style="list-style-type: none"> <li>• <b>f</b>: non-encryption cluster.</li> <li>• <b>t</b>: encryption cluster.</li> </ul>
g_tde_algo	text	Encryption algorithm. <ul style="list-style-type: none"> <li>• SM4-CTR-128</li> <li>• AES-CTR-128</li> </ul>
remain	text	Reserved column.

### 12.3.231 PG\_TIMEZONE\_ABBREVS

PG\_TIMEZONE\_ABBREVS displays information about all available time zones.

**Table 12-337** PG\_TIMEZONE\_ABBREVS columns

Name	Type	Description
abbrev	text	Abbreviation of the time zone name.
utc_offset	interval	Offset from UTC.
is_dst	Boolean	Specifies whether the DST is being used. If the DST is being used, the value is <b>TRUE</b> ; otherwise, the value is <b>FALSE</b> .

### 12.3.232 PG\_TIMEZONE\_NAMES

PG\_TIMEZONE\_NAMES displays all time zone names that can be recognized by SET TIMEZONE, along with their abbreviations, UTC offsets, and daylight saving time (DST) statuses.

**Table 12-338** PG\_TIMEZONE\_NAMES columns

Name	Type	Description
name	text	Name of the time zone.
abbrev	text	Abbreviation of the time zone name.
utc_offset	interval	Offset from UTC.

Name	Type	Description
is_dst	Boolean	Specifies whether the DST is being used. If the DST is being used, the value is <b>TRUE</b> ; otherwise, the value is <b>FALSE</b> .

### 12.3.233 PG\_TOTAL\_MEMORY\_DETAIL

**PG\_TOTAL\_MEMORY\_DETAIL** displays memory usage of a node in the database.

**Table 12-339** PG\_TOTAL\_MEMORY\_DETAIL columns

Name	Type	Description
nodename	text	Node name
memorytype	text	Memory name
memorybytes	integer	Size of the used memory, in MB

### 12.3.234 PG\_TOTAL\_USER\_RESOURCE\_INFO

**PG\_TOTAL\_USER\_RESOURCE\_INFO** displays resource usage of all users. Only administrators can query this view. This view is valid only when the GUC parameter **use\_workload\_manager** is set to **on**. I/O monitoring items are valid only when **enable\_logical\_io\_statistics** is set to **on**.

**Table 12-340** PG\_TOTAL\_USER\_RESOURCE\_INFO columns

Name	Type	Description
username	name	Username.
used_memory	integer	Used memory, in MB.
total_memory	integer	Available memory, in MB. The value <b>0</b> indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	double precision	Number of CPU cores in use. CPU usage data is collected only in complex jobs, and the value is the CPU usage of the related Cgroup.
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node.
used_space	bigint	Used permanent table storage space, in KB.

Name	Type	Description
total_space	bigint	Available permanent table storage space, in KB (-1 if the storage space is not limited).
used_temp_space	bigint	Used temporary space, in KB.
total_temp_space	bigint	Total available temporary space, in KB (-1 if the temporary space is not limited).
used_spill_space	bigint	Size of the used operator-level data flushing space, in KB.
total_spill_space	bigint	Total size of the available operator-level data flushing space, in KB (-1 if the space is not limited).
read_kbytes	bigint	CN: total bytes read by the user's complex jobs on all DNs in the last 5 seconds, in KB. DN: total bytes read by the user's complex jobs from the instance startup time to the current time, in KB.
write_kbytes	bigint	CN: total bytes written by the user's complex jobs on all DNs in the last 5 seconds, in KB. DN: total bytes written by the user's complex jobs from the instance startup time to the current time, in KB.
read_counts	bigint	CN: total number of read times of the user's complex jobs on all DNs in the last 5 seconds DN: total number of read times of the user's complex jobs from the instance startup time to the current time
write_counts	bigint	CN: total number of write times of the user's complex jobs on all DNs in the last 5 seconds DN: total number of write times of the user's complex jobs from the instance startup time to the current time
read_speed	double precision	CN: average read rate of the user's complex jobs on a single DN in the last 5 seconds, in KB/s. DN: average read rate of the user's complex jobs on the DN in the last 5 seconds, in KB/s.
write_speed	double precision	CN: average write rate of the user's complex jobs on a single DN in the last 5 seconds, in KB/s. DN: average write rate of the user's complex jobs on a single DN in the last 5 seconds, in KB/s.

### 12.3.235 PG\_TOTAL\_USER\_RESOURCE\_INFO\_OID

PG\_TOTAL\_USER\_RESOURCE\_INFO\_OID displays resource usage of all users. Only administrators can query this view. This view is valid only when the GUC parameter **use\_workload\_manager** is set to **on**.

**Table 12-341** PG\_TOTAL\_USER\_RESOURCE\_INFO\_OID columns

Name	Type	Description
userid	oid	User ID.
used_memory	integer	Used memory, in MB.
total_memory	integer	Available memory, in MB. The value <b>0</b> indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	double precision	Number of CPU cores in use.
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node.
used_space	bigint	Used storage space, in KB.
total_space	bigint	Available storage space, in KB (-1 if the space is not limited).
used_temp_space	bigint	Used temporary space, in KB.
total_temp_space	bigint	Total available temporary space, in KB (-1 if the temporary space is not limited).
used_spill_space	bigint	Used disk space for spilling, in KB.
total_spill_space	bigint	Total available disk space for spilling, in KB (-1 if the space is not limited).
read_kbytes	bigint	Volume of data read from the disk, in KB.
write_kbytes	bigint	Volume of data written to the disk, in KB.
read_counts	bigint	Number of disk read times.
write_counts	bigint	Number of disk write times.
read_speed	double precision	Disk read rate, in B/ms.
write_speed	double precision	Disk write rate, in B/ms.

## 12.3.236 PG\_USER

PG\_USER displays information about database users. By default, only the initial user and users with the sysadmin attribute can view the information. Other users can view the information only after being granted with permissions.

**Table 12-342** PG\_USER columns

Name	Type	Description
username	name	Username.
usesysid	oid	ID of this user.
usecreatedb	Boolean	Specifies whether a user has the permissions to create databases. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes.</li><li>• <b>f</b> (false): no.</li></ul>
usesuper	Boolean	Specifies whether the user is the initial system administrator with the highest permissions. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes.</li><li>• <b>f</b> (false): no.</li></ul>
usecatupd	Boolean	Specifies whether the user can directly update system catalogs. Only the initial system administrator whose <b>usesysid</b> is <b>10</b> has this permission. This permission is unavailable for other users. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes.</li><li>• <b>f</b> (false): no.</li></ul>
userepl	Boolean	Specifies whether the user has the permissions to duplicate data streams. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes.</li><li>• <b>f</b> (false): no.</li></ul>
passwd	text	Encrypted user password. The value is displayed as <b>*****</b> .
valbegin	timestamp with time zone	Start time for account validity ( <b>NULL</b> if start time is not specified).
valuntil	timestamp with time zone	End time for account validity ( <b>NULL</b> if end time is not specified).
respool	name	Resource pool where the user is in.
parent	oid	Parent user OID.

Name	Type	Description
spacelimit	text	Storage space of the permanent table, in KB.
useconfig	text[]	Default value of runtime configuration items.
tempspacelimit	text	Storage space of the temporary table, in KB.
spillspacelimit	text	Operator disk spill space, in KB.
usemonitoradmin	Boolean	Specifies whether the user is a monitor administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
useoperatoradmin	Boolean	Specifies whether the user is an O&M administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>
usepolicyadmin	Boolean	Specifies whether the user is a security policy administrator. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes.</li> <li>• <b>f</b> (false): no.</li> </ul>

### 12.3.237 PG\_USER\_MAPPINGS

PG\_USER\_MAPPINGS displays user mapping information. All users can view the report.

**Table 12-343** PG\_USER\_MAPPINGS columns

Name	Type	Reference	Description
umid	oid	<a href="#">PG_USER_MAPPING.oid</a>	OID of the user mapping.
srvid	oid	<a href="#">PG_FOREIGN_SERVER.oid</a>	OID of the foreign server that contains the mapping.
srvname	name	<a href="#">PG_FOREIGN_SERVER.srvname</a>	Name of the foreign server.
umuser	oid	<a href="#">PG_AUTHID.oid</a>	OID of the local role being mapped ( <b>0</b> if the user mapping is public)
username	name	-	Name of the local user to be mapped.

Name	Type	Reference	Description
umoptions	text[]	-	User mapping specific options. If the current user is the owner of the foreign server, it is a string in the format of "keyword=value". Otherwise, it is <b>null</b> .

### 12.3.238 PG\_VARIABLE\_INFO

PG\_VARIABLE\_INFO records information about transaction IDs and OIDs of the current node in a cluster.

**Table 12-344** PG\_VARIABLE\_INFO columns

Name	Type	Description
node_name	text	Node name.
next_oid	oid	OID generated next time on the node.
next_xid	xid	Transaction ID generated next time on the node.
oldest_xid	xid	Oldest transaction ID on the node.
xid_vac_limit	xid	Critical point (transaction ID) that triggers forcible autovacuum.
oldest_xid_db	oid	OID of the database that has the minimum datafrozenxid on the node.
last_extend_cs n_logpage	xid	Number of the last extended CSNlog page.
start_extend_cs n_logpage	xid	Number of the page from which CSNlog extending starts.
next_commit_s eqno	xid	CSN generated next time on the node.
latest_complet ed_xid	xid	Latest transaction ID on the node after the transaction commission or rollback.
startup_max_xi d	xid	Last transaction ID before the node is powered off.

### 12.3.239 PG\_VIEWS

PG\_VIEWS displays useful information about each view in the database.

**Table 12-345** PG\_VIEWS columns

Name	Type	Reference	Description
schemaname	name	nspname in <a href="#">PG_NAMESPACE</a>	Schema name of a view.
viewname	name	relname in <a href="#">PG_CLASS</a>	View name.
viewowner	name	Erolname in <a href="#">PG_AUTHID</a>	Owner of the view.
definition	text	-	Definition of the view.

### 12.3.240 PGXC\_COMM\_DELAY

**PGXC\_COMM\_DELAY** displays the communication library delay status of all DNs. Only system admin and monitor admin can view the status.

**Table 12-346** PGXC\_COMM\_DELAY columns

Name	Type	Description
node_name	text	Node name
remote_name	text	Peer node name
remote_host	text	IP address of the peer node
stream_num	integer	Number of logical stream connections used by the current physical connection
min_delay	integer	Minimum delay of the current physical connection within 1 minute, in microsecond <b>NOTE</b> A negative result is invalid. Wait until the delay status is updated and query again.
average	integer	Average delay of the current physical connection within 1 minute, in microsecond
max_delay	integer	Maximum delay of the current physical connection within 1 minute, in microsecond

### 12.3.241 PGXC\_COMM\_RECV\_STREAM

**PGXC\_COMM\_RECV\_STREAM** displays the receiving stream status of the communication libraries for all the DNs.

**Table 12-347** PGXC\_COMM\_RECV\_STREAM columns

Name	Type	Description
node_name	text	Node name

Name	Type	Description
local_tid	bigint	ID of the thread using this stream
remote_name	text	Peer node name
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Status of the stream <ul style="list-style-type: none"><li>● <b>UNKNOWN</b>: The logical connection status is unknown.</li><li>● <b>READY</b>: The logical connection is ready.</li><li>● <b>RUN</b>: The logical connection sends packets normally.</li><li>● <b>HOLD</b>: The logical connection is waiting to send packets.</li><li>● <b>CLOSED</b>: The logical connection is closed.</li><li>● <b>TO_CLOSED</b>: The logical connection will be closed.</li></ul>
query_id	bigint	<b>debug_query_id</b> corresponding to the stream
pn_id	integer	<b>plan_node_id</b> of the query executed by the stream
send_smp	integer	<b>smpid</b> of the sender of the query executed by the stream
recv_smp	integer	<b>smpid</b> of the receiver of the query executed by the stream
recv_bytes	bigint	Total data volume received from the stream, in byte
time	bigint	Current lifecycle service duration of the stream, in ms
speed	bigint	Average receiving rate of the stream, in byte/s
quota	bigint	Current communication quota value of the stream, in byte
buff_usize	bigint	Current size of the data cache of the stream, in byte

## 12.3.242 PGXC\_COMM\_SEND\_STREAM

**PGXC\_COMM\_SEND\_STREAM** displays the sending stream status of the communication libraries for all the DNs.

**Table 12-348** PGXC\_COMM\_SEND\_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Peer node name
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Status of the stream The options are as follows: <ul style="list-style-type: none"><li>● <b>UNKNOWN</b>: The connection status is unknown.</li><li>● <b>READY</b>: The connection is ready.</li><li>● <b>RUN</b>: The connection sends packets normally.</li><li>● <b>HOLD</b>: The connection is waiting to send packets.</li><li>● <b>CLOSED</b>: The connection is closed.</li><li>● <b>TO_CLOSED</b>: The connection will be closed.</li></ul>
query_id	bigint	<b>debug_query_id</b> corresponding to the stream
pn_id	integer	<b>plan_node_id</b> of the query executed by the stream
send_smp	integer	<b>smpid</b> of the sender of the query executed by the stream
recv_smp	integer	<b>smpid</b> of the receiver of the query executed by the stream
send_bytes	bigint	Total data volume sent by the stream, in byte
time	bigint	Current lifecycle service duration of the stream, in ms
speed	bigint	Average sending rate of the stream, in byte/s

Name	Type	Description
quota	bigint	Current communication quota value of the stream, in byte
wait_quota	bigint	Extra time generated when the stream waits the quota value, in ms

### 12.3.243 PGXC\_COMM\_STATUS

**PGXC\_COMM\_STATUS** displays the communication library status for all the DNs.

**Table 12-349** PGXC\_COMM\_STATUS columns

Name	Type	Description
node_name	text	Node name
rxpck_rate	integer	Receiving rate of the communication library on the node, in byte/s
txpck_rate	integer	Sending rate of the communication library on the node, in byte/s
rxkbyte_rate	bigint	Receiving rate of the communication library on the node, in kbyte/s
txkbyte_rate	bigint	Sending rate of the communication library on the node, in kbyte/s
buffer	bigint	Size of the buffer of the Cmailbox
memkbyte_libcomm	bigint	Communication memory size of the <b>libcomm</b> process, in bytes
memkbyte_libpq	bigint	Communication memory size of the <b>libpq</b> process, in bytes
used_pm	integer	Real-time usage of the postmaster thread
used_sflow	integer	Real-time usage of the <b>gs_sender_flow_controller</b> thread
used_rflow	integer	Real-time usage of the <b>gs_receiver_flow_controller</b> thread
used_rloop	integer	Highest real-time usage among multiple <b>gs_receivers_loop</b> threads
stream	integer	Total number of used logical connections.

## 12.3.244 PGXC\_GET\_STAT\_ALL\_TABLES

PGXC\_GET\_STAT\_ALL\_TABLES displays information about INSERTION, UPDATE, and DELETION operations on tables and the dirty page rate of tables. Before running **VACUUM FULL** to a system catalog with a high dirty page rate, ensure that no user is performing operations on it. This view is a new feature of GaussDB, and will not collect statistics on insertion, update, and deletion operations in earlier versions. You are advised to run **VACUUM FULL** to tables (excluding system catalogs) whose dirty page rate exceeds 30% or run it based on service scenarios. Only users with the system admin or monitor admin permission can view the information.

**Table 12-350** PGXC\_GET\_STAT\_ALL\_TABLES columns

Name	Type	Description
relid	oid	Table OID
relname	name	Table name.
schemaname	name	Name of the schema that contains a table.
n_tup_ins	numeric	Number of inserted tuples
n_tup_upd	numeric	Number of updated tuples
n_tup_del	numeric	Number of deleted tuples
n_live_tup	numeric	Number of live tuples
n_dead_tup	numeric	Number of dead tuples
dirty_page_rate	numeric(5,2)	Dirty page rate (%) of a table

## 12.3.245 PGXC\_GET\_TABLE\_SKEWNESS

PGXC\_GET\_TABLE\_SKEWNESS displays the data skew on tables in the current database. Only the system administrator has the permission to access.

**Table 12-351** PGXC\_GET\_TABLE\_SKEWNESS columns

Name	Type	Description
schemaname	name	Schema name of a table
tablename	name	Table name.
totalsize	numeric	Total size of the table, in bytes
avgsz	numeric(1000,0)	Average table size (total table size divided by the number of DN), which is the ideal size of tables distributed on each DN

Name	Type	Description
maxratio	numeric(4,3)	Ratio of the maximum table size on a single DN to the total table size
minratio	numeric(4,3)	Ratio of the minimum table size on a single DN to the total table size
skewsize	bigint	Table skew rate (the maximum table size on a single DN minus the minimum table size on a single DN)
skewratio	numeric(4,3)	Table skew rate (skew size divided by total table size)
skewstddev	numeric(1000,0)	Standard deviation of table distribution (For two tables of the same size, a larger deviation indicates a more severe skew.)

### 12.3.246 PGXC\_NODE\_ENV

PGXC\_NODE\_ENV displays environmental variables of all nodes in a cluster. This view can be viewed only by the users with monitor admin and sysadmin permission.

**Table 12-352** PGXC\_NODE\_ENV columns

Name	Type	Description
node_name1	text	Name of a node in a cluster.
host1	text	Host names of the nodes in the cluster.
process1	integer	Process IDs of the nodes in the cluster.
port1	integer	Port numbers of the nodes in the cluster.
installpath1	text	Installation directory of the nodes in the cluster.
datapath1	text	Data directory of the nodes in the cluster.
log_directory1	text	Log directory of the nodes in the cluster.

### 12.3.247 PGXC\_OS\_THREADS

PGXC\_OS\_THREADS displays thread status information under all normal nodes in the current cluster. Only system administrators and monitor administrators can access this view.

**Table 12-353** PGXC\_OS\_THREADS columns

Name	Type	Description
node_name	text	Names of the normal nodes in the cluster.
pid	bigint	IDs of running threads among the normal node processes in the current cluster.
lwpid	integer	ID of the lightweight thread corresponding to <b>pid</b> .
thread_name	text	Name of the thread corresponding to <b>pid</b> .
creation_time	timestamp with time zone	Creation time of the thread corresponding to <b>pid</b> .

### 12.3.248 PGXC\_PREPARED\_XACTS

PGXC\_PREPARED\_XACTS displays two-phase transactions in the prepared phase. Only users with the system admin or monitor admin permission can view the information.

**Table 12-354** PGXC\_PREPARED\_XACTS columns

Name	Type	Description
pgxc_prepared_xact	text	Displays the two-phase transaction in the prepared phase.

### 12.3.249 PGXC\_RUNNING\_XACTS

PGXC\_RUNNING\_XACTS displays information about running transactions on each node in the cluster. The content is the same as that displayed by PG\_RUNNING\_XACTS. Only users with the system admin or monitor admin permission can view the information.

**Table 12-355** PGXC\_RUNNING\_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM.
gxid	xid	Transaction ID.

Name	Type	Description
state	tinyint	Transaction status. <ul style="list-style-type: none"><li>• <b>3</b>: prepared.</li><li>• <b>0</b>: starting.</li></ul>
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	Boolean	Specifies whether the current transaction is lazy vacuum. <ul style="list-style-type: none"><li>• <b>t</b> (true): yes.</li><li>• <b>f</b> (false): no.</li></ul>
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the <b>prepared</b> state. If the status is not <b>prepared</b> , the value is <b>0</b> .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Minimum CSN of a local active transaction.

## 12.3.250 PGXC\_SQL\_COUNT

**PGXC\_SQL\_COUNT** displays node-level and user-level statistical results for the **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO** statements in real time, identifies query types with heavy load, and measures the capability of a cluster or node to perform a specific type of query. For example, you can calculate QPS based on the quantities of the five types of SQL statements at certain time points. **USER1 SELECT** is counted as **X1** at T1 and as **X2** at T2. The **SELECT** QPS of the user can be calculated as follows:  $(X2 - X1)/(T2 - T1)$ . In this way, the system can draw cluster-user-level QPS curve graphs and determine cluster throughput, tracing changes in the service load of each user. If there are drastic changes, the system can locate the specific statement type (such as **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**). You can also observe QPS curves to determine the time points when problems occur and then locate the problems using other tools. The curves provide a basis for optimizing cluster performance and locating problems. This view can be viewed only by the users with monitor admin and sysadmin permission. The query can be performed only on the CN. The **execute direct on (dn)'select \* from PGXC\_SQL\_COUNT'**; statement is not supported.

Columns in the **PGXC\_SQL\_COUNT** view are the same as those in the **GS\_SQL\_COUNT** view. For details, see [Table 12-227](#).

### NOTE

If a **MERGE INTO** statement can be pushed down and a DN receives it, the statement will be counted on the DN and the value of the **mergeinto\_count** column will be incremented by 1. If pushdown is not allowed, the DN will receive an **UPDATE** or **INSERT** statement. In this case, the **update\_count** or **insert\_count** column will be incremented by 1.

## 12.3.251 PGXC\_STAT\_ACTIVITY

PGXC\_STAT\_ACTIVITY displays information about the current user's queries on all CNs in the current cluster. Only SYSADMIN and MONADMIN can query the view.

**Table 12-356** PGXC\_STAT\_ACTIVITY columns

Name	Type	Description
coorname	text	Name of a CN in the current cluster.
datid	oid	OID of the database that the user session connects to in the backend.
datname	text	Name of the database that the user session connects to in the backend.
pid	bigint	Backend thread ID.
sessionid	bigint	Session ID.
usesysid	oid	OID of the user logged in to the backend.
username	text	Name of the user logged in to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is <b>null</b> , either the client is connected via a Unix socket on the server machine or this is an internal process such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (-1 if a Unix socket is used).
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server.
xact_start	timestamp with time zone	Time when the current transaction was started ( <b>null</b> if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the <b>query_start</b> column.

Name	Type	Description
query_start	timestamp with time zone	Time when the currently active query was started, or if <b>state</b> is not <b>active</b> , when the last query was started.
state_change	timestamp with time zone	Time when <b>state</b> was last modified.
waiting	Boolean	Specifies whether the backend is currently waiting for a lock. If the backend is currently waiting for a lock, the value is <b>true</b> . Otherwise, the value is <b>false</b> . If a lock is being waited for on a DN, determine the DN based on the <b>wait_status</b> column in the PG_THREAD_WAIT_STATUS view and query the <b>waiting</b> column in PG_STAT_ACTIVITY on the DN to check whether a lock wait occurs.
enqueue	text	Queuing status of a statement. Its value can be: <ul style="list-style-type: none"> <li>• <b>waiting in queue</b>: The statement is in the queue.</li> <li>• <b>Empty</b>: The statement is running.</li> </ul>

Name	Type	Description
state	text	<p>Overall status of this backend. Its value can be:</p> <ul style="list-style-type: none"> <li>• <b>active</b>: The backend is executing a query.</li> <li>• <b>idle</b>: The backend is waiting for a new client command.</li> <li>• <b>idle in transaction</b>: The backend is in a transaction, but there is no statement being executed in the transaction.</li> <li>• <b>idle in transaction (aborted)</b>: The backend is in a transaction, but there are statements failed in the transaction.</li> <li>• <b>fastpath function call</b>: The backend is executing a fast-path function.</li> <li>• <b>disabled</b>: This state is reported if <b>track_activities</b> is disabled in this backend.</li> </ul> <p><b>NOTE</b> Only system administrators can view the session status of their accounts. The state information of other accounts is empty. For example, after user <b>judy</b> is connected to the database, the state information of user <b>joe</b> and the initial user <b>omm</b> in <code>pgxc_stat_activity</code> is empty.</p> <pre>SELECT datname, username, usesysid, state,pid FROM pgxc_stat_activity;  datname   username   usesysid   state   pid -----+-----+-----+-----+----- +-----+ testdb   omm   10      139968752121616 testdb   omm   10      139968903116560 db_tpcds   judy   16398   active    139968391403280 testdb   omm   10      139968643069712 testdb   omm   10      139968680818448 testdb   joe   16390      139968563377936 (6 rows)</pre>
resource_pool	name	Resource pool used by the user.
query_id	bigint	ID of a query.
query	text	Latest query at the backend. If <b>state</b> is <b>active</b> , this column shows the ongoing query. In all other states, it shows the last query that was executed.

Name	Type	Description
connection_info	text	A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database. For details, see the GUC parameter <b>connection_info</b> .
global_sessionid	text	Global session ID.
unique_sql_id	bigint	Unique SQL statement ID.
trace_id	text	Driver-specific trace ID, which is associated with an application request.
top_xid	xid	Top-level transaction ID of a transaction.
current_xid	xid	Current transaction ID of a transaction.
xlog_quantity	bigint	Amount of Xlogs currently used by a transaction, in bytes.

### 12.3.252 PGXC\_STAT\_BAD\_BLOCK

PGXC\_STAT\_BAD\_BLOCK displays statistics about page verification failures after all nodes in a cluster are started. This view can be viewed only by the monitor admin and sysadmin users.

**Table 12-357** PGXC\_STAT\_BAD\_BLOCK columns

Name	Type	Description
nodename	text	Node name
databaseid	integer	Database OID
tablespaceid	integer	Tablespace OID
relfilenode	integer	File object ID
forknum	integer	File type
error_count	integer	Number of verification failures
first_time	timestamp with time zone	Time of the first verification failure
last_time	timestamp with time zone	Time of the latest verification failure

## 12.3.253 PGXC\_THREAD\_WAIT\_STATUS

PGXC\_THREAD\_WAIT\_STATUS displays all the call layer hierarchy relationship between threads of the SQL statements on all the nodes in a cluster, and the waiting status of the block for each thread, so that you can easily locate the causes of process response failures and similar phenomena.

The definitions of PGXC\_THREAD\_WAIT\_STATUS and PG\_THREAD\_WAIT\_STATUS are the same, because the essence of PGXC\_THREAD\_WAIT\_STATUS is the query summary of PG\_THREAD\_WAIT\_STATUS on each node in the cluster.

**Table 12-358** PGXC\_THREAD\_WAIT\_STATUS columns

Name	Type	Description
node_name	text	Current node name.
db_name	text	Database name.
thread_name	text	Thread name.
query_id	bigint	Query ID. It is equivalent to <b>debug_query_id</b> .
tid	bigint	Thread ID of the current thread.
sessionid	bigint	Session ID.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent session ID.
tlevel	integer	Level of the streaming thread.
smpid	integer	Concurrent thread ID.
wait_status	text	Detailed information about the waiting status of the current thread.
wait_event	text	Event that the current thread is waiting for. For details, see <a href="#">Table 12-331</a> .
locktag	text	Information about the lock that the current thread is waiting for.
lockmode	text	Lock mode that the current thread is waiting to obtain.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock.
global_sessionid	text	Global session ID.

Example:

If you run a statement on coordinator1 and no response is returned after a long period of time, establish another connection to coordinator1 to check the thread status on it.

```

gaussdb=# SELECT * FROM pg_thread_wait_status WHERE query_id > 0;
-[RECORD 1]-----+-----
node_name | cn_5001
db_name | tpcc_row
thread_name | PostgreSQL JDBC Driver
query_id | 72620544050065400
tid | 140650239031040
sessionid | 11680
lwtid | 26762
psessionid |
tlevel | 0
smpid | 0
wait_status | wait node: dn_6007_6008_6009, total 1
wait_event | wait node
locktag |
lockmode |
block_sessionid |
global_sessionid | 1120683504:11680#0
(1 rows)
gaussdb=# SELECT * FROM pgxc_thread_wait_status WHERE query_id > 0;
-[RECORD 1]-----+-----
node_name | cn_5001
db_name | tpcc_row
thread_name | PostgreSQL JDBC Driver
query_id | 72620544050081616
tid | 140648290055936
sessionid | 11680
lwtid | 26839
psessionid |
tlevel | 0
smpid | 0
wait_status | wait node: dn_6004_6005_6006, total 2
wait_event | wait node
locktag |
lockmode |
block_sessionid |
global_sessionid | 1120683504:11680#0
(1 rows)

```

### 12.3.254 PGXC\_TOTAL\_MEMORY\_DETAIL

PGXC\_TOTAL\_MEMORY\_DETAIL displays memory usage in the cluster. The query can be performed only on the CN. The **execute direct on (dn)'select \* from PGXC\_TOTAL\_MEMORY\_DETAIL'**; statement is not supported. Only users with the sysadmin and monitor admin permissions can query this view.

**Table 12-359** PGXC\_TOTAL\_MEMORY\_DETAIL columns

Name	Type	Description
nodename	text	Node name.

Name	Type	Description
memorytype	text	Memory name. <ul style="list-style-type: none"> <li>• <b>max_process_memory</b>: memory occupied by a GaussDB cluster instance.</li> <li>• <b>process_used_memory</b>: memory occupied by a GaussDB process.</li> <li>• <b>max_dynamic_memory</b>: maximum dynamic memory.</li> <li>• <b>dynamic_used_memory</b>: used dynamic memory.</li> <li>• <b>dynamic_peak_memory</b>: dynamic memory peak.</li> <li>• <b>dynamic_used_shrctx</b>: maximum dynamic shared memory context.</li> <li>• <b>dynamic_peak_shrctx</b>: dynamic peak value of the shared memory context.</li> <li>• <b>max_shared_memory</b>: maximum shared memory.</li> <li>• <b>shared_used_memory</b>: used shared memory.</li> <li>• <b>max_sctpcomm_memory</b>: maximum memory allowed for the communications library.</li> <li>• <b>sctpcomm_used_memory</b>: memory used by the communications library.</li> <li>• <b>sctpcomm_peak_memory</b>: memory peak of the communications library.</li> <li>• <b>other_used_memory</b>: other used memory.</li> </ul>
memorybytes	integer	Size of the used memory (unit: MB).

### 12.3.255 PGXC\_VARIABLE\_INFO

**PGXC\_VARIABLE\_INFO** records information about transaction IDs and OIDs of all nodes in a cluster. This view can be viewed only by the monitor admin and sysadmin users. The query can be performed only on the CN. The **execute direct on (dn)'select \* from PGXC\_VARIABLE\_INFO'**; statement is not supported.

**Table 12-360** PGXC\_VARIABLE\_INFO columns

Name	Type	Description
node_name	text	Node name

Name	Type	Description
next_oid	oid	OID generated next time for the node
next_xid	xid	Transaction ID generated next time for the node
oldest_xid	xid	Oldest transaction ID on the node
xid_vac_limit	xid	Critical point (transaction ID) that triggers forcible autovacuum
oldest_xid_db	oid	OID of the database that has the minimum datafrozenxid on the node
last_extend_csn_logpage	xid	Number of the last extended cslog page
start_extend_csn_logpage	xid	Number of the page from which cslog extending starts
next_commit_seqno	xid	CSN generated next time for the node
latest_completed_xid	xid	Latest transaction ID on the node after the transaction commission or rollback
startup_max_xid	xid	Last transaction ID before the node is powered off

## 12.3.256 PLAN\_TABLE

PLAN\_TABLE displays plan information collected by **EXPLAIN PLAN**. Plan information is in a session-level lifecycle. After a session exits, the data will be deleted. Data is isolated between sessions and between users. This view exists in both the PG\_CATALOG and SYS schemas.

**Table 12-361** PLAN\_TABLE columns

Name	Type	Description
statement_id	character varying(30)	Query tag specified by a user.
plan_id	bigint	Query ID.
id	integer	ID of each operator in a generated plan.
operation	character varying(30)	Operation description of an operator in a plan.
options	character varying(255)	Operation option.

Name	Type	Description
object_name	name	Object name corresponding to the operation, which is not the object alias used in the query. The object name is defined by users.
object_type	character varying(30)	Object type.
object_owner	name	Schema to which the object belongs. It is defined by users.
projection	character varying(4000)	Returned column information.
cost	double precision	Execution cost estimated by the optimizer for an operator.
cardinality	double precision	Number of rows estimated by the optimizer for an operator.
remarks	character varying(4000)	Not supported. Its value is <b>NULL</b> .
timestamp	date	Not supported. Its value is <b>NULL</b> .
object_node	character varying(128)	Not supported. Its value is <b>NULL</b> .
object_alias	character varying(261)	Not supported. Its value is <b>NULL</b> .
object_instance	numeric	Not supported. Its value is <b>NULL</b> .
optimizer	character varying(255)	Not supported. Its value is <b>NULL</b> .
search_columns	numeric	Not supported. Its value is <b>NULL</b> .
parent_id	numeric	Not supported. Its value is <b>NULL</b> .
depth	numeric	Not supported. Its value is <b>NULL</b> .
position	numeric	Not supported. Its value is <b>NULL</b> .
bytes	numeric	Not supported. Its value is <b>NULL</b> .
other_tag	character varying(255)	Not supported. Its value is <b>NULL</b> .
partition_start	character varying(255)	Not supported. Its value is <b>NULL</b> .
partition_stop	character varying(255)	Not supported. Its value is <b>NULL</b> .
partition_id	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
other	character varying	Not supported. Its value is <b>NULL</b> .
other_xml	clob	Not supported. Its value is <b>NULL</b> .
distribution	character varying(20)	Not supported. Its value is <b>NULL</b> .
cpu_cost	numeric	Not supported. Its value is <b>NULL</b> .
io_cost	numeric	Not supported. Its value is <b>NULL</b> .
temp_space	numeric	Not supported. Its value is <b>NULL</b> .
access_predicates	character varying(4000)	Not supported. Its value is <b>NULL</b> .
filter_predicates	character varying(4000)	Not supported. Its value is <b>NULL</b> .
time	numeric	Not supported. Its value is <b>NULL</b> .
qblock_name	character varying(128)	Not supported. Its value is <b>NULL</b> .

#### NOTE

- A valid **object\_type** value consists of a relkind type defined in **PG\_CLASS** (**TABLE**, **INDEX**, **SEQUENCE**, **VIEW**, or **TOASTVALUE TOAST**) and the rtekind type used in the plan (**SUBQUERY**, **JOIN**, **FUNCTION**, **VALUES**, **CTE**, or **REMOTE\_QUERY**).
- For RangeTableEntry (RTE), **object\_owner** is the object description used in the plan. Non-user-defined objects do not have **object\_owner**.
- Information in the **statement\_id**, **object\_name**, **object\_owner**, and **projection** columns is stored in letter cases specified by users and information in other columns is stored in uppercase.
- **PLAN\_TABLE** supports only **SELECT** and **DELETE** and does not support other DML operations.

## 12.3.257 PV\_FILE\_STAT

**PV\_FILE\_STAT** records statistics about data file I/O to indicate I/O performance and detect performance problems such as abnormal I/O operations.

**Table 12-362** PV\_FILE\_STAT columns

Name	Type	Description
filenum	oid	File ID
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	bigint	Number of times of reading physical files

Name	Type	Description
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks
readtim	bigint	Total duration of reading, in microseconds
writetim	bigint	Total duration of writing, in microseconds
avgiotim	bigint	Average duration of reading and writing, in microseconds
lstiotim	bigint	Duration of the last file reading, in microseconds
miniotim	bigint	Minimum duration of reading and writing, in microseconds
maxiowtm	bigint	Maximum duration of reading and writing, in microseconds

## 12.3.258 PV\_INSTANCE\_TIME

PV\_INSTANCE\_TIME records time consumption information of the current node. The time consumption information is classified into the following types:

- DB\_TIME: effective time spent by jobs in multi-core scenarios.
- CPU\_TIME: CPU time spent.
- EXECUTION\_TIME: time spent within executors.
- PARSE\_TIME: time spent on parsing SQL statements.
- PLAN\_TIME: time spent on generating plans.
- REWRITE\_TIME: time spent on rewriting SQL statements.
- PL\_EXECUTION\_TIME: execution time of the PL/SQL stored procedure
- PL\_COMPILATION\_TIME: compilation time of the PL/SQL stored procedure
- NET\_SEND\_TIME: time spent on the network.
- DATA\_IO\_TIME: I/O time spent.

**Table 12-363** PV\_INSTANCE\_TIME columns

Name	Type	Description
stat_id	integer	Statistics ID.
stat_name	text	Type name.

Name	Type	Description
value	bigint	Time value, in $\mu$ s.

### 12.3.259 PV\_OS\_RUN\_INFO

PV\_OS\_RUN\_INFO displays the running status of the OS.

Table 12-364 PV\_OS\_RUN\_INFO columns

Name	Type	Description
id	integer	ID
name	text	Name of the OS running status
value	numeric	Value of the OS running status
comments	text	Remarks of the OS running status
cumulative	boolean	Whether the value of the OS running status is cumulative

### 12.3.260 PV\_REDO\_STAT

PV\_REDO\_STAT displays the log replay of session threads.

Table 12-365 PV\_REDO\_STAT columns

Name	Type	Description
phywrts	bigint	Number of times that data is written during log replay.
phyblkwrt	bigint	Number of data blocks written during log replay.
writetim	bigint	Total time required for writing data during log replay.
avgiotim	bigint	Average time required for writing data during log replay.
lstiotim	bigint	Time consumed by the last data write operation during log replay.
miniotim	bigint	Minimum time consumed by a single data write operation during log replay.
maxiowtm	bigint	Maximum time consumed by a single data write operation during log replay.

## 12.3.261 PV\_SESSION\_MEMORY

PV\_SESSION\_MEMORY collects statistics about memory usage at the session level, including all the memory allocated to gaussdb and Stream threads on DNs for jobs currently executed by users.

**Table 12-366** PV\_SESSION\_MEMORY columns

Name	Type	Description
sessid	text	Thread start time and ID.
init_mem	integer	Memory allocated to the currently executed jobs before they enter the executor, in MB.
used_mem	integer	Memory allocated to the currently executed jobs, in MB.
peak_mem	integer	Peak memory allocated to the currently executed jobs, in MB.

## 12.3.262 PV\_SESSION\_MEMORY\_CONTEXT

PV\_SESSION\_MEMORY\_CONTEXT displays the memory usage of all sessions based on the MemoryContext node. This view is valid only when **enable\_thread\_pool** is set to **on**.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

**Table 12-367** PV\_SESSION\_MEMORY\_CONTEXT columns

Name	Type	Description
sessid	text	Session start time+session ID (string: <i>timestamp.sessionid</i> ).
threadid	bigint	ID of the thread bound to a session (-1 if no thread is bound).
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.
parent	text	Name of the parent memory context.
totalsize	bigint	Total size of the memory context, in bytes.

Name	Type	Description
freesize	bigint	Total size of released memory in the memory context, in bytes.
usedsize	bigint	Size of used memory in the memory context, in bytes. For <b>TempSmallContextGroup</b> , this parameter specifies the number of collected memory contexts.

---

**⚠ CAUTION**

This view is an O&M view and is used to locate memory problems. Do not query this view concurrently. If you query this view concurrently, the waiting time for new connections increases as the number of concurrent connections increases. As a result, new connections cannot be connected for a long time.

---

### 12.3.263 PV\_SESSION\_MEMORY\_DETAIL

PV\_SESSION\_MEMORY\_DETAIL displays the memory usage of sessions based on the MemoryContext node. When **enable\_thread\_pool** is set to **on**, this view contains memory usage of all threads and sessions.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

You can run the **SELECT \* FROM pv\_session\_memctx\_detail (threadid,');** statement to record information about all memory contexts of a thread into the *threadid\_timestamp.log* file in the *\$GAUSSLOG/pg\_log/\${node\_name}/dumppmem* directory. *threadid* can be obtained from **sessid** in the following table.

**Table 12-368** PV\_SESSION\_MEMORY\_DETAIL columns

Name	Type	Description
sessid	text	<ol style="list-style-type: none"> <li>1. When the thread pool is disabled (<b>enable_thread_pool</b> is set to <b>off</b>), this column indicates the thread start time +session ID (string: <i>timestamp.sessionid</i>).</li> <li>2. When the thread pool is enabled (<b>enable_thread_pool</b> is set to <b>on</b>): If the memory context is at the thread level, this column indicates the thread start time + thread ID (string: <i>timestamp.threadid</i>). If the memory context is at the session level, the column indicates the thread start time +session ID (string: <i>timestamp.sessionid</i>).</li> </ol>
sesstype	text	Thread name.
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.
parent	text	Name of the parent memory context.
totalsize	bigint	Total size of the memory context, in bytes.
freesize	bigint	Total size of released memory in the memory context, in bytes.
usedsize	bigint	Size of used memory in the memory context, in bytes. For <b>TempSmallContextGroup</b> , this parameter specifies the number of collected memory contexts.

 CAUTION

This view is an O&M view and is used to locate memory problems. Do not query this view concurrently. If you query this view concurrently, the waiting time for new connections increases as the number of concurrent connections increases. As a result, new connections cannot be connected for a long time.

### 12.3.264 PV\_SESSION\_STAT

PV\_SESSION\_STAT collects statistics about session states based on session threads or the AutoVacuum thread.

**Table 12-369** PV\_SESSION\_STAT columns

Name	Type	Description
sessid	text	Thread ID and start time.

Name	Type	Description
statid	integer	Statistics ID.
statname	text	Name of the statistics session.
statunit	text	Unit of the statistics session.
value	bigint	Value of the statistics session.

### 12.3.265 PV\_SESSION\_TIME

PV\_SESSION\_TIME displays the running time of session threads and time consumed in each execution phase.

**Table 12-370** PV\_SESSION\_TIME columns

Name	Type	Description
sessid	text	Thread ID and start time.
stat_id	integer	Statistics ID.
stat_name	text	Name of the session type. <ul style="list-style-type: none"> <li>DB_TIME: effective time spent by jobs in multi-core scenarios.</li> <li>CPU_TIME: CPU time spent.</li> <li>EXECUTION_TIME: time spent within executors.</li> <li>PARSE_TIME: time spent on parsing SQL statements.</li> <li>PLAN_TIME: time spent on generating plans.</li> <li>REWRITE_TIME: time spent on rewriting SQL statements.</li> <li>PL_EXECUTION_TIME: execution time of the PL/pgSQL stored procedure.</li> <li>PL_COMPILATION_TIME: compilation time of the PL/pgSQL stored procedure.</li> <li>NET_SEND_TIME: time spent on the network.</li> <li>DATA_IO_TIME: I/O time spent.</li> </ul>
value	bigint	Session value.

### 12.3.266 PV\_THREAD\_MEMORY\_CONTEXT

PV\_THREAD\_MEMORY\_CONTEXT displays the memory usage of all threads based on the MemoryContext node. This view is equivalent to the PV\_SESSION\_MEMORY\_DETAIL view when **enable\_thread\_pool** is set to **off**.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

**Table 12-371** PV\_THREAD\_MEMORY\_CONTEXT columns

Name	Type	Description
threadid	text	Thread start time and ID (string: <i>timestamp.sessionid</i> ).
tid	bigint	Thread ID.
thrdtype	text	Thread type. It can be any thread type in the system, such as wlmmonitor.
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.
parent	text	Name of the parent memory context.
totalsize	bigint	Total size of the memory context, in bytes.
freesize	bigint	Total size of released memory in the memory context, in bytes.
usedsize	bigint	Size of used memory in the memory context, in bytes. For <b>TempSmallContextGroup</b> , this parameter specifies the number of collected memory contexts.

## 12.3.267 PV\_TOTAL\_MEMORY\_DETAIL

PV\_TOTAL\_MEMORY\_DETAIL displays statistics about memory usage of the current database node in the unit of MB.

**Table 12-372** PV\_TOTAL\_MEMORY\_DETAIL columns

Name	Type	Description
nodename	text	Node name.

Name	Type	Description
memorytype	text	Memory type. The value must be one of the following: <ul style="list-style-type: none"> <li>● <b>max_process_memory</b>: memory occupied by a GaussDB cluster instance.</li> <li>● <b>process_used_memory</b>: memory occupied by a GaussDB process.</li> <li>● <b>max_dynamic_memory</b>: maximum dynamic memory.</li> <li>● <b>dynamic_used_memory</b>: used dynamic memory.</li> <li>● <b>dynamic_peak_memory</b>: dynamic memory peak.</li> <li>● <b>dynamic_used_shrctx</b>: maximum dynamic shared memory context.</li> <li>● <b>dynamic_peak_shrctx</b>: dynamic peak value of the shared memory context.</li> <li>● <b>max_backend_memory</b>: maximum memory that can be used when the HA port is used to execute services.</li> <li>● <b>backend_used_memory</b>: memory that has been used when the HA port is used to execute services.</li> <li>● <b>max_shared_memory</b>: maximum shared memory.</li> <li>● <b>shared_used_memory</b>: used shared memory.</li> <li>● <b>max_sctpcomm_memory</b>: maximum memory allowed for the communications library.</li> <li>● <b>sctpcomm_used_memory</b>: memory used by the communications library.</li> <li>● <b>sctpcomm_peak_memory</b>: memory peak of the communications library.</li> <li>● <b>other_used_memory</b>: other used memory.</li> </ul>
memorybytes	integer	Size of allocated memory-typed memory.

### 12.3.268 ROLE\_ROLE\_PRIVS

ROLE\_ROLE\_PRIVS displays roles granted to other roles and provides only information about the roles that the user has access to. By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-373** ROLE\_ROLE\_PRIVS columns

Name	Type	Description
role	character varying(128)	Role name.
granted_role	character varying(128)	Role to be granted.
admin_option	character varying(3)	Specifies whether the grant contains the <b>ADMIN</b> option. <ul style="list-style-type: none"> <li>• <b>YES</b></li> <li>• <b>NO</b></li> </ul>
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

## 12.3.269 ROLE\_SYS\_PRIVS

ROLE\_SYS\_PRIVS displays information about system privileges granted to roles (only roles accessible to the user are displayed). By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-374** ROLE\_SYS\_PRIVS columns

Name	Type	Description
role	character varying(128)	Role name.
privilege	character varying(40)	System permission granted to a role.  System permissions include rolsuper, rolinherit, rolcreatorole, rolcreatedb, rolcatupdate, rolcanlogin, rolreplication, rolauditadmin, rolsystemadmin, roluseft, rolmonitoradmin, roloperatoradmin, and rolpolicyadmin.
admin_option	character varying(3)	Specifies whether the grant contains the <b>ADMIN</b> option. <ul style="list-style-type: none"> <li>• <b>YES</b></li> <li>• <b>NO</b></li> </ul>
common	character varying(3)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.270 ROLE\_TAB\_PRIVS

ROLE\_TAB\_PRIVS displays information about object permissions granted to roles (only roles accessible to the user are displayed). By default, it is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-375** ROLE\_TAB\_PRIVS columns

Name	Type	Description
role	character varying(128)	Role name.
owner	character varying(128)	Object owner.
table_name	character varying(128)	Name of an object. Object types include tables, packages, indexes, and sequences.
column_name	character varying(128)	Not supported. Its value is <b>NULL</b> .
privilege	character varying(40)	Permissions on an object, including USAGE, UPDATE, DELETE, INSERT, CONNECT, SELECT, and EXECUTE.
grantable	character varying(3)	Specifies whether the grant contains the <b>GRANT</b> option. <ul style="list-style-type: none"> <li>• <b>YES</b></li> <li>• <b>NO</b></li> </ul>
common	character varying(3)	Not supported. Its value is <b>NULL</b> .
inherited	character varying(3)	Not supported. Its value is <b>NULL</b> .

### 12.3.271 SYS\_DUMMY

SYS\_DUMMY is automatically created by the database based on the data dictionary. It has only one text column in only one row for storing expression calculation results. This view is accessible to all users. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-376** SYS\_DUMMY columns

Name	Type	Description
dummy	text	Expression calculation result

## 12.3.272 V\_INSTANCE

V\_INSTANCE displays instance information in current database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-377** V\_INSTANCE columns

Name	Type	Description
instance_number	oid	OID of the current database.
instance_name	character varying(16)	Name of the current database.
host_name	character varying(64)	Host name.
version	character varying(17)	Not supported. Its value is <b>NULL</b> .
version_legacy	character varying(17)	Not supported. Its value is <b>NULL</b> .
version_full	character varying(17)	Not supported. Its value is <b>NULL</b> .
startup_time	date	Not supported. Its value is <b>NULL</b> .
status	character varying(12)	Not supported. Its value is <b>NULL</b> .
parallel	character varying(3)	Not supported. Its value is <b>NULL</b> .
thread#	numeric	Not supported. Its value is <b>NULL</b> .
archiver	character varying(7)	Not supported. Its value is <b>NULL</b> .
log_switch_wait	character varying(15)	Not supported. Its value is <b>NULL</b> .
logins	character varying(10)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
shutdown_pending	character varying(3)	Not supported. Its value is <b>NULL</b> .
database_status	character varying(17)	Not supported. Its value is <b>NULL</b> .
instance_role	character varying(18)	Not supported. Its value is <b>NULL</b> .
active_state	character varying(9)	Not supported. Its value is <b>NULL</b> .
blocked	character varying(3)	Not supported. Its value is <b>NULL</b> .
con_id	numeric	Not supported. Its value is <b>NULL</b> .
instance_mode	character varying(11)	Not supported. Its value is <b>NULL</b> .
edition	character varying(7)	Not supported. Its value is <b>NULL</b> .
family	character varying(80)	Not supported. Its value is <b>NULL</b> .
database_type	character varying(15)	Not supported. Its value is <b>NULL</b> .

## 12.3.273 V\_MYSTAT

V\_MYSTAT displays statistics about all sessions in the database. Only the system administrator can access this view. Common users can access this view only after being authorized. This view exists in both PG\_CATALOG and SYS schema.

**Table 12-378** V\_MYSTAT columns

Name	Type	Description
sid	numeric	Current session ID.
statistic#	numeric	Not supported. Its value is <b>NULL</b> .
value	numeric	Not supported. Its value is <b>NULL</b> .
con_id	numeric	Not supported. Its value is <b>NULL</b> .

## 12.3.274 V\_SESSION

V\_SESSION displays information about all current sessions. Only administrators can access this view. Common users can access this view only after being authorized. This view exists in both PG\_CATALOG and SYS schemas.

Table 12-379 V\_SESSION columns

Name	Type	Description
saddr	raw	Not supported. Its value is <b>NULL</b> .
sid	bigint	Session ID.
serial#	integer	Sequence number of the active backend thread, which is <b>0</b> in GaussDB.
audsid	numeric	Not supported. Its value is <b>NULL</b> .
paddr	raw	Not supported. Its value is <b>NULL</b> .
schema#	numeric	Not supported. Its value is <b>NULL</b> .
schemaname	name	Name of the user logged in to the backend.
user#	oid	OID of the user logged in to the backend thread. Its value is <b>0</b> if the backend thread is a global auxiliary thread.
username	name	Name of the user logged in to the backend thread. <b>username</b> is null if the backend thread is a global auxiliary thread.
command	numeric	Not supported. Its value is <b>NULL</b> .
ownerid	numeric	Not supported. Its value is <b>NULL</b> .
taddr	character varying(16)	Not supported. Its value is <b>NULL</b> .
lockwait	character varying(16)	Not supported. Its value is <b>NULL</b> .
machine	text	Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.
sql_id	bigint	ID of a query.
client_info	text	Client information.
event	text	Queuing status of a statement. Its value can be: <ul style="list-style-type: none"><li>• <b>waiting in queue</b>: The statement is in the queue.</li><li>• <b>Empty</b>: The statement is running.</li></ul>

Name	Type	Description
sql_exec_start	timestamp with time zone	Time when the current active query was started, or time when the last query was started if the value of <b>state</b> is not <b>active</b> .
program	text	Name of the application connected to the backend.
status	text	Overall status of this backend. Its value can be: <ul style="list-style-type: none"> <li>● <b>active</b>: The backend is executing a query.</li> <li>● <b>idle</b>: The backend is waiting for a new client command.</li> <li>● <b>idle in transaction</b>: The backend is in a transaction, but there is no statement being executed in the transaction.</li> <li>● <b>idle in transaction (aborted)</b>: The backend is in a transaction, but there are statements failed in the transaction.</li> <li>● <b>fastpath function call</b>: The backend is executing a fast-path function.</li> <li>● <b>disabled</b>: This state is reported if <b>track_activities</b> is disabled in this backend.</li> </ul>
server	character varying(9)	Not supported. Its value is <b>NULL</b> .
pdml_status	character varying(8)	Specifies whether to enable a DML parallel execution in the current session.
port	numeric	Port number of the current session.
process	character varying(24)	Process ID of the current session.
logon_time	date	Login time of the current session.
last_call_et	integer	Duration when the status of the current session changes last time.
osuser	character varying(128)	Not supported. Its value is <b>NULL</b> .
terminal	character varying(30)	Not supported. Its value is <b>NULL</b> .
type	character varying(10)	Not supported. Its value is <b>NULL</b> .
sql_address	raw	Not supported. Its value is <b>NULL</b> .
sql_hash_value	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
sql_child_number	numeric	Not supported. Its value is <b>NULL</b> .
sql_exec_id	numeric	Not supported. Its value is <b>NULL</b> .
prev_sql_addr	raw	Not supported. Its value is <b>NULL</b> .
prev_hash_value	numeric	Not supported. Its value is <b>NULL</b> .
prev_sql_id	character varying(13)	Not supported. Its value is <b>NULL</b> .
prev_child_number	numeric	Not supported. Its value is <b>NULL</b> .
prev_exec_start	date	Not supported. Its value is <b>NULL</b> .
prev_exec_id	numeric	Not supported. Its value is <b>NULL</b> .
plsql_entry_object_id	numeric	Not supported. Its value is <b>NULL</b> .
plsql_entry_subprogram_id	numeric	Not supported. Its value is <b>NULL</b> .
plsql_object_id	numeric	Not supported. Its value is <b>NULL</b> .
plsql_subprogram_id	numeric	Not supported. Its value is <b>NULL</b> .
module	character varying(64)	Not supported. Its value is <b>NULL</b> .
module_hash	numeric	Not supported. Its value is <b>NULL</b> .
action	character varying(64)	Not supported. Its value is <b>NULL</b> .
action_hash	numeric	Not supported. Its value is <b>NULL</b> .
fixed_table_sequence	numeric	Not supported. Its value is <b>NULL</b> .
row_wait_obj#	numeric	Not supported. Its value is <b>NULL</b> .
row_wait_file#	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
row_wait_block#	numeric	Not supported. Its value is <b>NULL</b> .
row_wait_row#	numeric	Not supported. Its value is <b>NULL</b> .
top_level_call#	numeric	Not supported. Its value is <b>NULL</b> .
pdml_enabled	character varying(3)	Not supported. Its value is <b>NULL</b> .
failover_type	character varying(13)	Not supported. Its value is <b>NULL</b> .
failover_method	character varying(10)	Not supported. Its value is <b>NULL</b> .
failed_over	character varying(3)	Not supported. Its value is <b>NULL</b> .
resource_consumer_group	character varying(32)	Not supported. Its value is <b>NULL</b> .
pddl_status	character varying(8)	Not supported. Its value is <b>NULL</b> .
pq_status	character varying(8)	Not supported. Its value is <b>NULL</b> .
current_queue_duration	numeric	Not supported. Its value is <b>NULL</b> .
client_identifier	character varying(64)	Not supported. Its value is <b>NULL</b> .
blocking_session_status	character varying(11)	Not supported. Its value is <b>NULL</b> .
blocking_instance	numeric	Not supported. Its value is <b>NULL</b> .
blocking_session	numeric	Not supported. Its value is <b>NULL</b> .
final_blocking_session_status	character varying(11)	Not supported. Its value is <b>NULL</b> .
final_blocking_instance	numeric	Not supported. Its value is <b>NULL</b> .
final_blocking_session	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
seq#	numeric	Not supported. Its value is <b>NULL</b> .
event#	numeric	Not supported. Its value is <b>NULL</b> .
p1text	character varying(64)	Not supported. Its value is <b>NULL</b> .
p1	numeric	Not supported. Its value is <b>NULL</b> .
p1raw	raw	Not supported. Its value is <b>NULL</b> .
p2text	character varying(64)	Not supported. Its value is <b>NULL</b> .
p2	numeric	Not supported. Its value is <b>NULL</b> .
p2raw	raw	Not supported. Its value is <b>NULL</b> .
p3text	character varying(64)	Not supported. Its value is <b>NULL</b> .
p3	numeric	Not supported. Its value is <b>NULL</b> .
p3raw	raw	Not supported. Its value is <b>NULL</b> .
wait_class_id	numeric	Not supported. Its value is <b>NULL</b> .
wait_class#	numeric	Not supported. Its value is <b>NULL</b> .
wait_class	character varying(64)	Not supported. Its value is <b>NULL</b> .
wait_time	numeric	Not supported. Its value is <b>NULL</b> .
seconds_in_wait	numeric	Not supported. Its value is <b>NULL</b> .
state	character varying(19)	Not supported. Its value is <b>NULL</b> .
wait_time_micro	numeric	Not supported. Its value is <b>NULL</b> .
time_remaining_micro	numeric	Not supported. Its value is <b>NULL</b> .
time_since_last_wait_micro	numeric	Not supported. Its value is <b>NULL</b> .
service_name	character varying(64)	Not supported. Its value is <b>NULL</b> .
sql_trace	character varying(8)	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
sql_trace_waits	character varying(5)	Not supported. Its value is <b>NULL</b> .
sql_trace_binids	character varying(5)	Not supported. Its value is <b>NULL</b> .
sql_trace_plan_stats	character varying(10)	Not supported. Its value is <b>NULL</b> .
session_edition_id	numeric	Not supported. Its value is <b>NULL</b> .
creator_addr	raw	Not supported. Its value is <b>NULL</b> .
creator_serial#	numeric	Not supported. Its value is <b>NULL</b> .
ecid	character varying(64)	Not supported. Its value is <b>NULL</b> .
sql_translation_profile_id	numeric	Not supported. Its value is <b>NULL</b> .
pga_tunable_mem	numeric	Not supported. Its value is <b>NULL</b> .
shard_ddl_statuses	character varying(8)	Not supported. Its value is <b>NULL</b> .
con_id	numeric	Not supported. Its value is <b>NULL</b> .
external_name	character varying(1024)	Not supported. Its value is <b>NULL</b> .
plsql_debugger_connected	character varying(5)	Not supported. Its value is <b>NULL</b> .

### 12.3.275 V\$NLS\_PARAMETERS

V\$NLS\_PARAMETERS displays the National Language Support (NLS) parameters and parameter values configured for the database. This view exists in both the PG\_CATALOG and SYS schema and all users can access this view.

**Table 12-380** V\$NLS\_PARAMETERS columns

Name	Type	Description
parameter	character varying(64)	NLS parameter name.

Name	Type	Description
value	character varying(64)	NLS parameter value.
con_id	numeric	Not supported. Its value is <b>0</b> .

## 12.3.276 V\$SESSION\_WAIT

V\$SESSION\_WAIT displays the current wait event or the last wait event of each session of each user. By default, only system administrators can access this view. Common users can access the view only after being authorized. This view exists in both the PG\_CATALOG and SYS schemas.

**Table 12-381** V\$SESSION\_WAIT columns

Name	Type	Description
sid	numeric	Session ID, which is mapped to the <b>V\$SESSION.SID</b> column.
seq#	numeric	Not supported. Its value is <b>NULL</b> .
event	character varying(64 )	If the session is waiting, the resource or event that is waiting for is displayed. If the session is not waiting, the resource or event that is waiting for the last time is displayed.
p1text	character varying(64 )	Not supported. Its value is <b>NULL</b> .
p1	numeric	Not supported. Its value is <b>NULL</b> .
p1raw	raw	Not supported. Its value is <b>NULL</b> .
p2text	character varying(64 )	Not supported. Its value is <b>NULL</b> .
p2	numeric	Not supported. Its value is <b>NULL</b> .
p2raw	raw	Not supported. Its value is <b>NULL</b> .
p3text	character varying(64 )	Not supported. Its value is <b>NULL</b> .
p3	numeric	Not supported. Its value is <b>NULL</b> .
p3raw	raw	Not supported. Its value is <b>NULL</b> .
wait_class_id	numeric	Not supported. Its value is <b>NULL</b> .
wait_class#	numeric	Not supported. Its value is <b>NULL</b> .

Name	Type	Description
wait_class	character varying(64)	Name of a wait event type
wait_time	numeric	<p>If the session is waiting, the value is <b>0</b>. If the session is not waiting, the options are as follows:</p> <ul style="list-style-type: none"> <li>• <b>&gt;0</b>: duration of the last wait (unit: centisecond).</li> <li>• <b>-1</b>: The duration of the last wait is less than a centisecond.</li> <li>• <b>-2</b>: The <b>TIMED_STATISTICS</b> parameter is set to <b>false</b>.</li> </ul> <p>This column has been deprecated and replaced by <b>WAIT_TIME_MICRO</b> and <b>STATE</b>.</p>
second_in_wait	numeric	<p>If the session is waiting, the value is the amount of time to wait. If the session is not waiting, the value is the amount of time since the last wait started.</p> <p>This column has been deprecated and replaced by the <b>WAIT_TIME_MICRO</b> and <b>TIME_SINCE_LAST_WAIT_MICRO</b> columns.</p>
state	character varying(64)	<p>Waiting state. The options are as follows:</p> <ul style="list-style-type: none"> <li>• <b>WAITING</b>: The session is waiting.</li> <li>• <b>WAITED UNKNOWN TIME</b>: The duration of the last wait is unknown. This value is used when <b>TIMED_STATISTICS</b> is set to <b>false</b>.</li> <li>• <b>WAITED SHORT TIME</b>: The duration of the last wait is less than a centisecond.</li> <li>• <b>WAITED KNOWN TIME</b>: duration of the last wait specified in the <b>WAIT_TIME</b> column.</li> </ul>
wait_time_micro	numeric	Wait time, in microseconds. If the session is waiting, the value is the time spent in the current wait. If the session is not waiting, the value is the time spent for the last wait.
time_remaining_micro	numeric	Not supported. Its value is <b>NULL</b> .
time_since_last_wait_micro	numeric	Time elapsed since the last wait ended, in microseconds. If the session is waiting, the value is <b>0</b> .
con_id	numeric	Not supported. Its value is <b>0</b> .

## 12.3.277 V\$SYSSTAT

V\$SYSSTAT displays the resource usage of the entire database instance since the database instance is started. By default, only the initial user or monitor administrator can access the database. Other users can access the database only after being granted the MONADMIN permission. This view exists in both PG\_CATALOG and SYS schema. Contact an administrator to enable the memory resource management function in the Arm environment.

**Table 12-382** V\$SYSSTAT columns

Name	Type	Description
statistic#	numeric	Statistics ID.
name	character varying(64)	Statistical item name.
class	numeric	Not supported. Its value is <b>NULL</b> .
value	numeric	Statistical item value.
stat_id	numeric	Not supported. Its value is <b>NULL</b> .
con_id	numeric	Not supported. Its value is <b>0</b> .

## 12.3.278 V\$SYSTEM\_EVENT

V\$SYSTEM\_EVENT displays information about all the wait events (summary of each wait event since the instance is started). By default, only system administrators can access this view. Common users can access the view only after being authorized. This view exists in both the PG\_CATALOG and SYS schemas.

**Table 12-383** V\$SYSTEM\_EVENT columns

Name	Type	Description
event	character varying(64)	Wait event name
total_waits	numeric	Total number of wait events
total_timeouts	numeric	Total number of timeout events
time_waited	numeric	Total time (in centiseconds) of wait events
average_wait	numeric	Average time (in centiseconds) of wait events
time_waited_micro	numeric	Total time (in microseconds) of wait events

Name	Type	Description
total_waits_fg	numeric	Not supported. Its value is <b>NULL</b> .
total_timeouts_fg	numeric	Not supported. Its value is <b>NULL</b> .
time_waited_fg	numeric	Not supported. Its value is <b>NULL</b> .
average_wait_fg	numeric	Not supported. Its value is <b>NULL</b> .
time_waited_micro_fg	numeric	Not supported. Its value is <b>NULL</b> .
event_id	numeric	Not supported. Its value is <b>NULL</b> .
wait_class_id	numeric	Not supported. Its value is <b>NULL</b> .
wait_class#	numeric	Not supported. Its value is <b>NULL</b> .
wait_class	character varying(64)	Class name of a wait event
con_id	numeric	Not supported. Its value is <b>0</b> .

### 12.3.279 V\$VERSION

V\$VERSION displays the version number of the database. This view exists in the PG\_CATALOG and SYS schemas and all users can access this view.

**Table 12-384** V\$VERSION columns

Name	Type	Description
banner	character varying(80)	Component name and version number.
banner_full	character varying(160)	Database version and specific version number.
banner_legacy	character varying(80)	Database version.
con_id	numeric	Not supported. Its value is <b>0</b> .

# 13 Schemas

The following table describes the schemas supported in GaussDB.

 **NOTE**

Do not create service data of users in a schema that provides functional APIs, including but not limited to tables and functions (such as `dbe_*` and `pkg_*`).

**Table 13-1** Schemas supported in GaussDB

Schema	Description
dbe_perf	Diagnoses performance issues and is also the data source of WDR snapshots. After a database is installed, only the initial user and monitor administrator have permission to view views and functions in this schema by default.
snapshot	Manages data related to WDR snapshots. By default, the initial user or monitor administrator can access the data.
sqladvsior	Recommends distribution keys. For details, see <a href="#">Distribution Key Recommendation Functions</a> .
sys	Provides the system information view APIs.
pg_catalog	Maintains system catalog information, including system catalogs and all built-in data types, functions, and operators.
pg_toast	Stores large objects (for internal use).
public	Public schema, which is used to store public objects. If <b>search_path</b> is not specified and a schema with the same name exists, new tables (and other objects) are created in the schema by default. If no schema with the same name exists, new tables (and other objects) are automatically placed in this public schema.
dbe_raw	Advanced function package <code>dbe_raw</code> , which is used to convert raw data, obtain substrings, and calculate the length.

Schema	Description
dbe_session	Advanced function package dbe_session, which is used to set the value of a specified attribute and support user query and verification.
dbe_lob	Advanced function package dbe_lob, which is used to read, write, and copy large files (CLOB/BLOB).
dbe_match	Advanced function package dbe_match, which is used to compare character string similarity.
dbe_task	Advanced function package dbe_task, which is used to schedule job tasks, including committing tasks, canceling tasks, synchronizing task status, and updating task information, so that the database can periodically execute specific tasks.
dbe_sql	Advanced function package dbe_sql, which is used to execute dynamic SQL statements and construct query and other commands during application running.
dbe_file	Advanced function package dbe_file, which is used to read, copy, write, delete, and rename external database files.
dbe_output	Advanced function package dbe_output, which is used to print output information.
dbe_random	Advanced function package dbe_random, which is used to generate random seeds and random numbers.
dbe_application_info	Advanced function package dbe_application_info, which is used for recording client information.
dbe_utility	Advanced function package dbe_utility, which is used to call the debugging tool in a stored procedure, for example, to print error stacks.
dbe_scheduler	Advanced function package dbe_scheduler, which is used to create scheduled jobs and enable the database to periodically execute specified tasks through programs and schedules. You can also perform external database tasks by authorizing and providing certificates.
information_schema	Stores information about objects defined in the current database.
dbe_sql_util	SQL O&M function, including the O&M API of SQL patches.

**Table 13-2** Schemas disabled in GaussDB

Schema	Description
dbe_pldebugger	Debugs PL/SQL functions and stored procedures. Currently, this view is not supported. An error message "unsupported" is displayed when the API is called in this view.
db4ai	Manages data of different versions in AI training.
dbe_pldeveloper	Compiles and debugs user stored procedures.

## 13.1 Information Schema

An information schema named INFORMATION\_SCHEMA automatically exists in all databases. An information schema consists of a group of views that contain information about objects defined in the current database. The owner of this schema is the initial database user. However, all users have only the permission to use this schema and do not have the permission to create objects such as tables and functions.

Information schemas are compatible with PG, including constraint\_table\_usage, domain\_constraints, domain\_udt\_usage, domains, enabled\_roles, key\_column\_usage, parameters, referential\_constraints, applicable\_roles, administrable\_role\_authorizations, attributes, character\_sets, check\_constraint\_routine\_usage, check\_constraints, collations, collation\_character\_set\_applicability, column\_domain\_usage, column\_privileges, column\_udt\_usage, columns, constraint\_column\_usage, role\_column\_grants, routine\_privileges, role\_routine\_grants, routines, schemata, sequences, table\_constraints, table\_privileges, role\_table\_grants, tables, triggered\_update\_columns, triggers, udt\_privileges, role\_udt\_grants, usage\_privileges, role\_usage\_grants, user\_defined\_types, view\_column\_usage, view\_routine\_usage, view\_table\_usage, views, data\_type\_privileges, element\_types, column\_options, foreign\_data\_wrapper\_options, foreign\_data\_wrappers, foreign\_server\_options, foreign\_servers, foreign\_table\_options, foreign\_tables, user\_mapping\_options, user\_mappings, sql\_features, sql\_implementation\_info, sql\_languages, sql\_packages, sql\_parts, sql\_sizing, and sql\_sizing\_profiles.

The following sections display only the views that are not listed in the preceding description.

### 13.1.1 \_PG\_FOREIGN\_DATA\_WRAPPERS

**\_PG\_FOREIGN\_DATA\_WRAPPERS** displays information about a foreign-data wrapper. Only the sysadmin user has the permission to view this view.

**Table 13-3** \_PG\_FOREIGN\_DATA\_WRAPPERS columns

Name	Type	Description
------	------	-------------

oid	oid	OID of the foreign-data wrapper
fdwowner	oid	OID of the owner of the foreign-data wrapper
fdwoptions	text[]	Foreign-data wrapper specific option, expressed in a string in the format of <i>keyword=value</i>
foreign_data_wrapper_catalog	information_schema.sql_identifier	Name of the database where the foreign-data wrapper is located (always the current database)
foreign_data_wrapper_name	information_schema.sql_identifier	Name of the foreign-data wrapper
authorization_identifier	information_schema.sql_identifier	Role of the owner of the foreign-data wrapper
foreign_data_wrapper_language	information_schema.character_data	Programming language of the foreign-data wrapper

## 13.1.2 \_PG\_FOREIGN\_SERVERS

**\_PG\_FOREIGN\_SERVERS** displays information about a foreign server. Only the sysadmin user has the permission to view this view.

**Table 13-4** \_PG\_FOREIGN\_SERVERS columns

Name	Type	Description
oid	oid	OID of the foreign server
srvoptions	text[]	Foreign server specific options, expressed in a string in the format of <i>keyword=value</i>
foreign_server_catalog	information_schema.sql_identifier	Name of the database where the foreign server is located (always the current database)
foreign_server_name	information_schema.sql_identifier	Name of the foreign server
foreign_data_wrapper_catalog	information_schema.sql_identifier	Name of the database where the foreign-data wrapper is located (always the current database)

foreign_data_wrapper_name	information_schema.sql_identifier	Name of the foreign-data wrapper
foreign_server_type	information_schema.character_data	Type of the foreign server
foreign_server_version	information_schema.character_data	Version of the foreign server
authorization_identifier	information_schema.sql_identifier	Role of the owner of the foreign server

### 13.1.3 \_PG\_FOREIGN\_TABLE\_COLUMNS

**\_PG\_FOREIGN\_TABLE\_COLUMNS** displays column information about a foreign table. Only the sysadmin user has the permission to view this view.

**Table 13-5** \_PG\_FOREIGN\_TABLE\_COLUMNS columns

Name	Type	Description
nspname	name	Schema name
relname	name	Table name
attname	name	Column name
attdwoptions	text[]	Attribute-level foreign data wrapper options, expressed in a string in the format of <i>keyword=value</i>

### 13.1.4 \_PG\_FOREIGN\_TABLES

**\_PG\_FOREIGN\_TABLES** stores information about all foreign tables defined in the current database, whereas displays information about foreign tables accessible to the current user. Only the sysadmin user has the permission to view this view.

**Table 13-6** \_PG\_FOREIGN\_TABLES columns

Name	Type	Description
foreign_table_catalog	information_schema.sql_identifier	Name of the database where the foreign table is located (always the current database)
foreign_table_schema	name	Name of the schema that the foreign table is in
foreign_table_name	name	Name of the foreign table

ftoptions	text[]	Foreign table options
foreign_server_catalog	information_schema.sql_identifier	Name of the database where the foreign server is located (always the current database)
foreign_server_name	information_schema.sql_identifier	Name of the foreign server
authorization_identifier	information_schema.sql_identifier	Role of the owner

### 13.1.5 \_PG\_USER\_MAPPINGS

Stores mappings from local users to remote users. Only the sysadmin user has the permission to view this view.

**Table 13-7** \_PG\_USER\_MAPPINGS columns

Name	Type	Description
oid	oid	OID of the mapping from the local user to a remote user.
umoptions	text[]	User mapping specific options, expressed in a string in the format of <i>keyword=value</i> .
umuser	oid	OID of the local user being mapped ( <b>0</b> if the user mapping is public).
authorization_identifier	information_schema.sql_identifier	Role of the local user.
foreign_server_catalog	information_schema.sql_identifier	Name of the database where the foreign server is located (always the current database).
foreign_server_name	information_schema.sql_identifier	Name of the foreign server.
srvowner	information_schema.sql_identifier	Owner of the foreign server.

### 13.1.6 INFORMATION\_SCHEMA\_CATALOG\_NAME

Displays the name of the current database.

**Table 13-8** INFORMATION\_SCHEMA\_CATALOG\_NAME columns

Name	Type	Description
catalog_name	information_schema.sql_identifier	Current database name.

## 13.2 DBE\_PERF Schema

In the **DBE\_PERF** schema, views are used to diagnose performance issues and are also the data source of WDR snapshots. After a database is installed, only the initial user and monitoring administrator have permission to view views and functions in the **DBE\_PERF** scheme by default. To ensure forward compatibility, the permission on the dbe\_perf schema has no change before and after an upgrade. In the current version, all users are not allowed to create operators in this schema. Existing operators are not affected. Organization views are divided based on multiple dimensions, such as OS, instance, and memory. These views comply with the following naming rules:

- A view starting with **GLOBAL\_** requests data from a CN/DN and returns the data without processing the data.
- A view starting with **SUMMARY\_** summarizes data in a cluster. In most cases, data from a CN/DN (sometimes only a CN) is processed, aggregated, and returned.
- A view that does not start with **GLOBAL\_** or **SUMMARY\_** is a local view and does not request data from any other CN/DN.

### 13.2.1 OS

#### 13.2.1.1 OS\_RUNTIME

**OS\_RUNTIME** displays the running status of the current OS.

**Table 13-9** OS\_RUNTIME columns

Name	Type	Description
id	integer	ID
name	text	Name of the OS running status
value	numeric	Value of the OS running status
comments	text	Remarks of the OS running status
cumulative	boolean	Whether the value of the OS running status is cumulative

### 13.2.1.2 GLOBAL\_OS\_RUNTIME

**GLOBAL\_OS\_RUNTIME** records the OS running status information on all normal nodes in the cluster.

**Table 13-10** GLOBAL\_OS\_RUNTIME columns

Name	Type	Description
node_name	name	Node name
id	integer	ID
name	text	Name of the OS running status
value	numeric	Value of the OS running status
comments	text	Remarks of the OS running status
cumulative	boolean	Whether the value of the OS running status is cumulative

### 13.2.1.3 OS\_THREADS

**OS\_THREADS** provides status information about all threads on the current node.

**Table 13-11** OS\_THREADS columns

Name	Type	Description
node_name	text	Current node name
pid	bigint	ID of the thread running within the current node process
lwpid	integer	Lightweight thread ID corresponding to <b>pid</b>
thread_name	text	Name of the thread corresponding to <b>pid</b>
creation_time	timestamp with time zone	Creation time of the thread corresponding to <b>pid</b>

### 13.2.1.4 GLOBAL\_OS\_THREADS

**GLOBAL\_OS\_THREADS** records the thread status information on all normal nodes in the cluster.

**Table 13-12** GLOBAL\_OS\_THREADS columns

Name	Type	Description
node_name	text	Current node name
pid	bigint	ID of the thread running within the current node process
lwpid	integer	Lightweight thread ID corresponding to <b>pid</b>
thread_name	text	Name of the thread corresponding to <b>pid</b>
creation_time	timestamp with time zone	Creation time of the thread corresponding to <b>pid</b>

## 13.2.2 Instance

### 13.2.2.1 INSTANCE\_TIME

Records the time consumption information on the current node. The information is classified into the following types:

- DB\_TIME: effective time spent by jobs in multi-core scenarios.
- CPU\_TIME: CPU time cost.
- EXECUTION\_TIME: time spent in the executor.
- PARSE\_TIME: time spent on parsing SQL statements.
- PLAN\_TIME: time spent on generating plans.
- REWRITE\_TIME: time spent on rewriting SQL statements.
- PL\_EXECUTION\_TIME: execution time of the PL/SQL stored procedure.
- PL\_COMPILATION\_TIME: compilation time of the PL/SQL stored procedure.
- NET\_SEND\_TIME: time spent on the network.
- DATA\_IO\_TIME: I/O time spent.

**Table 13-13** INSTANCE\_TIME columns

Name	Type	Description
stat_id	integer	Statistics ID.
stat_name	text	Type name.
value	bigint	Time value (unit: μs).

### 13.2.2.2 GLOBAL\_INSTANCE\_TIME

**GLOBAL\_INSTANCE\_TIME** records the time consumption information on all normal nodes in the cluster. For details about the time types, see the **INSTANCE\_TIME** view.

**Table 13-14** GLOBAL\_INSTANCE\_TIME columns

Name	Type	Description
node_name	name	Node name
stat_id	integer	Statistics ID
stat_name	text	Type name
value	bigint	Time value (unit: $\mu$ s)

## 13.2.3 Memory

### 13.2.3.1 MEMORY\_NODE\_DETAIL

Displays memory usage of the current database node.

**Table 13-15** MEMORY\_NODE\_DETAIL columns

Name	Type	Description
nodename	text	Node name.

Name	Type	Description
memorytype	text	Memory name. <ul style="list-style-type: none"> <li>● <b>max_process_memory</b>: maximum available memory of the database node.</li> <li>● <b>process_used_memory</b>: memory occupied by a process.</li> <li>● <b>max_dynamic_memory</b>: maximum dynamic memory.</li> <li>● <b>dynamic_used_memory</b>: used dynamic memory.</li> <li>● <b>dynamic_peak_memory</b>: dynamic memory peak.</li> <li>● <b>dynamic_used_shrctx</b>: context of the used dynamic shared memory.</li> <li>● <b>dynamic_peak_shrctx</b>: dynamic peak value of the shared memory context.</li> <li>● <b>max_shared_memory</b>: maximum shared memory.</li> <li>● <b>shared_used_memory</b>: used shared memory.</li> <li>● <b>max_sctpcomm_memory</b>: maximum memory allowed for TCP proxy communications.</li> <li>● <b>sctpcomm_used_memory</b>: used memory for TCP proxy communications.</li> <li>● <b>sctpcomm_peak_memory</b>: memory peak of TCP proxy communications.</li> <li>● <b>other_used_memory</b>: other used memory.</li> <li>● <b>gpu_max_dynamic_memory</b>: maximum dynamic GPU memory.</li> <li>● <b>gpu_dynamic_used_memory</b>: used dynamic GPU memory.</li> <li>● <b>gpu_dynamic_peak_memory</b>: dynamic GPU memory peak.</li> <li>● <b>pooler_conn_memory</b>: memory allocated to the connection pool.</li> <li>● <b>pooler_freeconn_memory</b>: memory occupied by idle connections in the connection pool.</li> <li>● <b>storage_compress_memory</b>: memory used by the storage module for compression.</li> <li>● <b>udf_reserved_memory</b>: reserved memory for the UDF.</li> </ul>
memorybytes	integer	Size of the used memory (unit: MB).

### 13.2.3.2 GLOBAL\_MEMORY\_NODE\_DETAIL

Displays the memory usage on all normal nodes in the cluster.

**Table 13-16** GLOBAL\_MEMORY\_NODE\_DETAIL columns

Name	Type	Description
nodename	text	Node name.
memorytype	text	<p>Memory name.</p> <ul style="list-style-type: none"> <li>● <b>max_process_memory</b>: maximum available memory of the database node.</li> <li>● <b>process_used_memory</b>: memory occupied by a process.</li> <li>● <b>max_dynamic_memory</b>: maximum dynamic memory.</li> <li>● <b>dynamic_used_memory</b>: used dynamic memory.</li> <li>● <b>dynamic_peak_memory</b>: dynamic memory peak.</li> <li>● <b>dynamic_used_shrctx</b>: context of the used dynamic shared memory.</li> <li>● <b>dynamic_peak_shrctx</b>: dynamic peak value of the shared memory context.</li> <li>● <b>max_shared_memory</b>: maximum shared memory.</li> <li>● <b>shared_used_memory</b>: used shared memory.</li> <li>● <b>max_sctpcomm_memory</b>: maximum memory allowed for TCP proxy communications.</li> <li>● <b>sctpcomm_used_memory</b>: used memory for TCP proxy communications.</li> <li>● <b>sctpcomm_peak_memory</b>: memory peak of TCP proxy communications.</li> <li>● <b>other_used_memory</b>: other used memory.</li> <li>● <b>gpu_max_dynamic_memory</b>: maximum dynamic GPU memory.</li> <li>● <b>gpu_dynamic_used_memory</b>: used dynamic GPU memory.</li> <li>● <b>gpu_dynamic_peak_memory</b>: dynamic GPU memory peak.</li> <li>● <b>pooler_conn_memory</b>: memory allocated to the connection pool.</li> <li>● <b>pooler_freeconn_memory</b>: memory occupied by idle connections in the connection pool.</li> <li>● <b>storage_compress_memory</b>: memory used by the storage module for compression.</li> <li>● <b>udf_reserved_memory</b>: reserved memory for the user-defined functions.</li> </ul>
memorybytes	integer	Size of the used memory (unit: MB).

### 13.2.3.3 MEMORY\_NODE\_NG\_DETAIL

**MEMORY\_NODE\_NG\_DETAIL** displays memory usage of a node group.

**Table 13-17** MEMORY\_NODE\_NG\_DETAIL columns

Name	Type	Description
ngname	text	Node group name
memorytype	text	Memory name <ul style="list-style-type: none"> <li>● <b>ng_total_memory</b>: total memory configured in the node group</li> <li>● <b>ng_used_memory</b>: used memory</li> <li>● <b>ng_estimate_memory</b>: memory used by the optimizer for evaluation</li> <li>● <b>ng_foreignrp_memsize</b>: memory configured in the foreign resource pool</li> <li>● <b>ng_foreignrp_usesize</b>: memory used by the foreign resource pool</li> <li>● <b>ng_foreignrp_peaksize</b>: peak memory used by the foreign resource pool</li> <li>● <b>ng_foreignrp_mempct</b>: percentage of the system memory configured in attributes of the foreign resource pool</li> <li>● <b>ng_foreignrp_estmsize</b>: memory used by the optimizer for job evaluation in the foreign resource pool</li> </ul>
memorybytes	integer	Size of the used memory (unit: MB)

### 13.2.3.4 SHARED\_MEMORY\_DETAIL

**SHARED\_MEMORY\_DETAIL** displays the usage information about shared memory contexts on the current node.

**Table 13-18** SHARED\_MEMORY\_DETAIL columns

Name	Type	Description
contextname	text	Name of the memory context
level	smallint	Level of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the shared memory (unit: byte)

Name	Type	Description
freesize	bigint	Remaining size of the shared memory (unit: byte)
usedsize	bigint	Used size of the shared memory (unit: byte)

### 13.2.3.5 GLOBAL\_SHARED\_MEMORY\_DETAIL

**GLOBAL\_SHARED\_MEMORY\_DETAIL** displays the usage information about shared memory contexts on all normal nodes in the cluster.

**Table 13-19** GLOBAL\_SHARED\_MEMORY\_DETAIL columns

Name	Type	Description
node_name	name	Node name
contextname	text	Name of the memory context
level	smallint	Level of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the shared memory (unit: byte)
freesize	bigint	Remaining size of the shared memory (unit: byte)
usedsize	bigint	Used size of the shared memory (unit: byte)

### 13.2.3.6 TRACK\_MEMORY\_CONTEXT\_DETAIL

Queries the detailed memory application information about the memory context set by **DBE\_PERF.track\_memory\_context**. Only the initial user or a user with the monadmin permission can execute this view.

**Table 13-20** TRACK\_MEMORY\_CONTEXT\_DETAIL columns

Name	Type	Description
context_name	text	Name of the memory context.
file	text	File to which the memory application location belongs.

Name	Type	Description
line	integer	Line number of the memory application location.
size	bigint	Total size of memory that is applied for (unit: byte).

## 13.2.4 File

### 13.2.4.1 FILE\_IOSTAT

Records statistics about data file I/Os to indicate I/O performance and detect performance problems such as abnormal I/O operations.

**Table 13-21** FILE\_IOSTAT columns

Name	Type	Description
filenum	oid	File identifier
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks
readtim	bigint	Total duration of reading (unit: $\mu$ s)
writetim	bigint	Total duration of writing (unit: $\mu$ s)
avgiotim	bigint	Average duration of reading and writing (unit: $\mu$ s)
lstiotim	bigint	Duration of the last file reading (unit: $\mu$ s)
miniotim	bigint	Minimum duration of reading and writing (unit: $\mu$ s)
maxiowtm	bigint	Maximum duration of reading and writing (unit: $\mu$ s)

### 13.2.4.2 SUMMARY\_FILE\_IOSTAT

Records the summary of statistics about data file I/Os of each node in the cluster to reflect performance issues such as exceptions in I/O operations.

The values of **phyrds**, **phywrts**, **phyblkrd**, **phyblkwrt**, **readtim** and **writetim** are summed up based on the data of each node. **avgiotim** is the average value (total duration/total times) of each node. **lstiotim** and **maxiowtm** are the maximum values of each node. **miniotim** is the minimum value of each node.

**Table 13-22** SUMMARY\_FILE\_IOSTAT columns

Name	Type	Description
filenum	oid	File ID.
dbid	oid	Database ID.
spcid	oid	Tablespace ID.
phyrds	numeric	Number of times of reading physical files.
phywrts	numeric	Number of times of writing into physical files.
phyblkrd	numeric	Number of times of reading physical file blocks.
phyblkwrt	numeric	Number of times of writing into physical file blocks.
readtim	numeric	Total duration of reading (unit: $\mu$ s).
writetim	numeric	Total duration of writing (unit: $\mu$ s).
avgiotim	bigint	Average duration of reading and writing (unit: $\mu$ s).
lstiotim	bigint	Duration of the last file reading (unit: $\mu$ s).
miniotim	bigint	Minimum duration of reading and writing (unit: $\mu$ s).
maxiowtm	bigint	Maximum duration of reading and writing (unit: $\mu$ s).

### 13.2.4.3 GLOBAL\_FILE\_IOSTAT

Displays statistics about data file I/Os on all nodes in the cluster.

**Table 13-23** GLOBAL\_FILE\_IOSTAT columns

Name	Type	Description
node_name	name	Node name.
filenum	oid	File ID.
dbid	oid	Database ID.
spcid	oid	Tablespace ID.
phyrds	bigint	Number of times of reading physical files.
phywrts	bigint	Number of times of writing into physical files.
phyblkrd	bigint	Number of times of reading physical file blocks.
phyblkwrt	bigint	Number of times of writing into physical file blocks.
readtim	bigint	Total duration of reading (unit: $\mu$ s).
writetim	bigint	Total duration of writing (unit: $\mu$ s).
avgiotim	bigint	Average duration of reading and writing (unit: $\mu$ s).
lstiotim	bigint	Duration of the last file reading (unit: $\mu$ s).
miniotim	bigint	Minimum duration of reading and writing (unit: $\mu$ s).
maxiowtm	bigint	Maximum duration of reading and writing (unit: $\mu$ s).

#### 13.2.4.4 FILE\_REDO\_IOSTAT

Records statistics about redo logs (WALs) on the current node.

**Table 13-24** FILE\_REDO\_IOSTAT columns

Name	Type	Description
phywrts	bigint	Number of times writing into the WAL buffer.
phyblkwrt	bigint	Number of blocks written into the WAL buffer.
writetim	bigint	Duration of writing into Xlog files (unit: $\mu$ s).

Name	Type	Description
avgiotim	bigint	Average duration of writing into Xlog files (unit: $\mu$ s). The value is <b>wrietim</b> divided by <b>phywrts</b> .
lstiotim	bigint	Duration of the last writing into Xlog files (unit: $\mu$ s).
miniotim	bigint	Minimum duration of writing into Xlog files (unit: $\mu$ s).
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: $\mu$ s).

### 13.2.4.5 SUMMARY\_FILE\_REDO\_IOSTAT

Displays statistics about redo logs (WALs) on all nodes in the cluster. The values of the **phywrts**, **phyblkwrt**, and **wrietim** columns are accumulated based on the data of each node. The value of **avgiotim** is the average value of each node (summed up **wrietim** or **phywrts**). The values of **lstiotim** and **maxiowtm** are the maximum values of each node, and the value of **miniotim** is the minimum value of each node.

Table 13-25 SUMMARY\_FILE\_REDO\_IOSTAT columns

Name	Type	Description
phywrts	numeric	Number of times writing into the WAL buffer.
phyblkwrt	numeric	Number of blocks written into the WAL buffer.
wrietim	numeric	Duration of writing into Xlog files (unit: $\mu$ s).
avgiotim	bigint	Average duration of writing into Xlog files (unit: $\mu$ s). The value is <b>wrietim</b> divided by <b>phywrts</b> .
lstiotim	bigint	Duration of the last writing into Xlog files (unit: $\mu$ s).
miniotim	bigint	Minimum duration of writing into Xlog files (unit: $\mu$ s).
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: $\mu$ s).

### 13.2.4.6 GLOBAL\_FILE\_REDO\_IOSTAT

Displays statistics about redo logs (WALs) on nodes in the cluster.

**Table 13-26** GLOBAL\_FILE\_REDO\_IOSTAT columns

Name	Type	Description
node_name	name	Node name.
phywrts	bigint	Number of times writing into the WAL buffer.
phyblkwrt	bigint	Number of blocks written into the WAL buffer.
writetim	bigint	Duration of writing into Xlog files (unit: $\mu$ s).
avgiotim	bigint	Average duration of writing into Xlog files (unit: $\mu$ s). The value is <b>writetim</b> divided by <b>phywrts</b> .
lstiotim	bigint	Duration of the last writing into Xlog files (unit: $\mu$ s).
miniotim	bigint	Minimum duration of writing into Xlog files (unit: $\mu$ s).
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: $\mu$ s).

### 13.2.4.7 LOCAL\_REL\_IOSTAT

Displays the accumulated I/O status of all data files on the current node.

**Table 13-27** LOCAL\_REL\_IOSTAT columns

Name	Type	Description
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks

### 13.2.4.8 GLOBAL\_REL\_IOSTAT

Displays statistics about data file I/Os on all nodes.

**Table 13-28** GLOBAL\_REL\_IOSTAT columns

Name	Type	Description
node_name	name	Node name
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks

### 13.2.4.9 SUMMARY\_REL\_IOSTAT

Obtains the summary of data file I/O statistics of all nodes.

**Table 13-29** SUMMARY\_REL\_IOSTAT columns

Name	Type	Description
phyrds	numeric	Number of times of reading physical files.
phywrts	numeric	Number of times of writing into physical files.
phyblkrd	numeric	Number of times of reading physical file blocks.
phyblkwrt	numeric	Number of times of writing into physical file blocks.

## 13.2.5 Object

### 13.2.5.1 STAT\_USER\_TABLES

Displays the status information about user-defined ordinary tables in all schemas on the current node.

**Table 13-30** STAT\_USER\_TABLES columns

Name	Type	Description
relid	oid	Table OID

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting <b>VACUUM FULL</b> )
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting <b>VACUUM FULL</b> )
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times this table has been manually analyzed

Name	Type	Description
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon thread.

### 13.2.5.2 SUMMARY\_STAT\_USER\_TABLES

Displays the sum of the status information about user-defined ordinary tables in all schemas of a cluster. It is used on the CN. The status information about user-defined common tables on each DN is summed up. The timestamp column is not summed up and only the latest value of this column on all nodes is used.

**Table 13-31** SUMMARY\_STAT\_USER\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on the table
seq_tup_read	numeric	Number of live rows fetched by sequential scans
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting <b>VACUUM FULL</b> )
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.

Name	Type	Description
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	numeric	Number of times this table has been manually vacuumed (not counting <b>VACUUM FULL</b> )
autovacuum_count	numeric	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	numeric	Number of times this table has been manually analyzed
autoanalyze_count	numeric	Number of times the table has been analyzed by the autovacuum daemon thread.

### 13.2.5.3 GLOBAL\_STAT\_USER\_TABLES

Displays the status of user-defined ordinary tables in all schemas on each node. It is used on CNs. The status information about user-defined ordinary tables on each DN is displayed but not summed up.

**Table 13-32** GLOBAL\_STAT\_USER\_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.

Name	Type	Description
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of dead rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (excluding VACUUM FULL).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times this table is manually vacuumed (excluding VACUUM FULL).
autovacuum_count	bigint	Number of times this table is vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times this table is manually analyzed.
autoanalyze_count	bigint	Number of times the table is analyzed by the autovacuum daemon thread.

### 13.2.5.4 STAT\_USER\_INDEXES

Displays the index status information about the user-defined ordinary tables on the current node.

**Table 13-33** STAT\_USER\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index.

Name	Type	Description
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.5 SUMMARY\_STAT\_USER\_INDEXES

Displays the index status statistics of user-defined ordinary tables in all schemas of a cluster. It applies to CNs only and the index status information about user-defined ordinary tables on each CN is summed up.

**Table 13-34** SUMMARY\_STAT\_USER\_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	numeric	Number of index scans performed on the index.
idx_tup_read	numeric	Number of index entries returned by scans on the index.
idx_tup_fetch	numeric	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.6 GLOBAL\_STAT\_USER\_INDEXES

Displays the index status information about user-defined ordinary tables in all schemas on each node. The information is about CNs and DNs. It is used on CNs and information is not summed up.

**Table 13-35** GLOBAL\_STAT\_USER\_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.7 STAT\_SYS\_TABLES

Displays status information about all the system catalogs in the pg\_catalog, information\_schema, and pg\_toast schemas on the current node.

**Table 13-36** STAT\_SYS\_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema where the table is located.
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted

Name	Type	Description
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting <b>VACUUM FULL</b> )
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting <b>VACUUM FULL</b> )
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times this table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon thread.

### 13.2.5.8 SUMMARY\_STAT\_SYS\_TABLES

Displays the status statistics of all system catalogs in the pg\_catalog, information\_schema, and pg\_toast schemas in the cluster. The statistics are used on CNs and include data on CNs and DN. The status information in the system catalog on each node is summed up. The timestamp column is not summed up and only the latest value of this column on all nodes is used.

**Table 13-37** SUMMARY\_STAT\_SYS\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on the table
seq_tup_read	numeric	Number of live rows fetched by sequential scans
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting <b>VACUUM FULL</b> )
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	numeric	Number of times this table has been manually vacuumed (not counting <b>VACUUM FULL</b> )
autovacuum_count	numeric	Number of times this table has been vacuumed by the autovacuum daemon

Name	Type	Description
analyze_count	numeric	Number of times this table has been manually analyzed
autoanalyze_count	numeric	Number of times the table has been analyzed by the autovacuum daemon thread.

### 13.2.5.9 GLOBAL\_STAT\_SYS\_TABLES

Displays the status information of all system catalogs in pg\_catalog, information\_schema, and pg\_toast modes on each node in the cluster. The statistics information includes that of CNs and DN. It is used on CNs and is not summed up.

**Table 13-38** GLOBAL\_STAT\_SYS\_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema where the table is located.
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of inactive rows.

Name	Type	Description
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting <b>VACUUM FULL</b> )
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting <b>VACUUM FULL</b> )
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times this table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon thread.

### 13.2.5.10 STAT\_SYS\_INDEXES

Displays the index status information about all system catalogs in the pg\_catalog, information\_schema, and pg\_toast schemas on the current node.

**Table 13-39** STAT\_SYS\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans performed on the index.

Name	Type	Description
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.11 SUMMARY\_STAT\_SYS\_INDEXES

Displays the index status statistics of all system catalogs in the `pg_catalog`, `information_schema`, and `pg_toast` schemas of all nodes in the cluster. It is used on CNs and the index status information of system catalogs on each node is summed up.

**Table 13-40** SUMMARY\_STAT\_SYS\_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	numeric	Number of index scans performed on the index.
idx_tup_read	numeric	Number of index entries returned by scans on the index.
idx_tup_fetch	numeric	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.12 GLOBAL\_STAT\_SYS\_INDEXES

Displays the index status information of all system catalogs in `pg_catalog`, `information_schema`, and `pg_toast` schemas on each node. The index status information is about CNs and DN. It is used on CNs and information is not summed up.

**Table 13-41** GLOBAL\_STAT\_SYS\_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.13 STAT\_ALL\_TABLES

Displays the status of each table (including the TOAST table) on the current node of the database.

**Table 13-42** STAT\_ALL\_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema where the table is located.
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

Name	Type	Description
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting <b>VACUUM FULL</b> )
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting <b>VACUUM FULL</b> )
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times this table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon thread.

### 13.2.5.14 SUMMARY\_STAT\_ALL\_TABLES

Displays the sum of the status information of each table (including the TOAST table) in the cluster database. It is used on the CN. The status information in the table on each node is summed up. The timestamp column is not summed up and only the latest value of this column on all nodes is used.

**Table 13-43** SUMMARY\_STAT\_ALL\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on the table
seq_tup_read	numeric	Number of live rows fetched by sequential scans

Name	Type	Description
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting <b>VACUUM FULL</b> )
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	numeric	Number of times this table has been manually vacuumed (not counting <b>VACUUM FULL</b> )
autovacuum_count	numeric	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	numeric	Number of times this table has been manually analyzed
autoanalyze_count	numeric	Number of times the table has been analyzed by the autovacuum daemon thread.

### 13.2.5.15 GLOBAL\_STAT\_ALL\_TABLES

Displays the status of each table (including the TOAST table) on each node. It is used on CNs and the status of each table on each node is not summed up.

**Table 13-44** GLOBAL\_STAT\_ALL\_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema where the table is located.
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting <b>VACUUM FULL</b> )
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting <b>VACUUM FULL</b> )
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon

Name	Type	Description
analyze_count	bigint	Number of times this table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon thread.

### 13.2.5.16 STAT\_ALL\_INDEXES

Displays the access information about each index on the current database node.

**Table 13-45** STAT\_ALL\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.17 SUMMARY\_STAT\_ALL\_INDEXES

Displays the access statistics of each index on each database node in a cluster. It is used on CNs. The access statistics of each node index are summed up.

**Table 13-46** SUMMARY\_STAT\_ALL\_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.

Name	Type	Description
idx_scan	numeric	Number of index scans performed on the index.
idx_tup_read	numeric	Number of index entries returned by scans on the index.
idx_tup_fetch	numeric	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.18 GLOBAL\_STAT\_ALL\_INDEXES

Displays the access information about each index on each node of the database in the cluster. It is used on the CN. The status information of each index on each node is not summed up.

**Table 13-47** GLOBAL\_STAT\_ALL\_INDEXES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	OID of the table for this index.
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

### 13.2.5.19 STAT\_DATABASE

Displays the statistics of the current node in the database.

**Table 13-48** STAT\_DATABASE columns

Name	Type	Description
datid	oid	Database OID.

Name	Type	Description
datname	name	Database name.
numbackends	integer	Number of backends currently connected to this database.
xact_committed	bigint	Number of transactions in this database that have been committed.
xact_rollback	bigint	Number of transactions in this database that have been rolled back.
blks_read	bigint	Number of disk blocks read in this database.
blks_hit	bigint	Number of disk blocks that have been hit in the cache. In this case, data does not need to be read from disks. (The cache includes only the buffer cache and does not include the file system cache of the OS.)
tup_returned	bigint	Number of live rows fetched by sequential scans and number of index rows returned by index scans in the database.
tup_fetched	bigint	Number of rows returned by the current database through indexes.
tup_inserted	bigint	Number of rows inserted.
tup_updated	bigint	Number of rows updated.
tup_deleted	bigint	Number of rows deleted.
conflicts	bigint	Number of queries canceled due to conflicts with database replay (conflicts occur only on the standby node). For details, see <a href="#">STAT_DATABASE_CONFLICTS</a> .
temp_files	bigint	Number of temporary files created by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter <b>log_temp_files</b> .
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter <b>log_temp_files</b> .
deadlocks	bigint	Number of deadlocks detected in this database.
blk_read_time	double precision	Time spent in reading data file blocks by backends in this database (unit: ms).

Name	Type	Description
blk_write_time	double precision	Time spent in writing data file blocks by backends in this database (unit: ms).
stats_reset	timestamp with time zone	Time at which the current statistics were reset.

### 13.2.5.20 SUMMARY\_STAT\_DATABASE

Displays the summary of the statistics on each database node in the cluster. It is used on CNs. The status information of each node in the database is summed up by database name. The timestamp column is not summed up and only the latest value of this column on all nodes is used.

**Table 13-49** SUMMARY\_STAT\_DATABASE columns

Name	Type	Description
datname	name	Database name.
numbackends	bigint	Number of backends currently connected to this database.
xact_commit	numeric	Number of transactions in this database that have been committed.
xact_rollback	numeric	Number of transactions in this database that have been rolled back.
blks_read	numeric	Number of disk blocks read in this database.
blks_hit	numeric	Number of disk blocks that have been hit in the cache. In this case, data does not need to be read from disks. (The cache includes only the buffer cache and does not include the file system cache of the OS.)
tup_returned	numeric	Number of live rows fetched by sequential scans and number of index rows returned by index scans in the database.
tup_fetched	numeric	Number of rows returned by the current database through indexes.
tup_inserted	bigint	Number of rows inserted.
tup_updated	bigint	Number of rows updated.
tup_deleted	bigint	Number of rows deleted.

Name	Type	Description
conflicts	bigint	Number of queries canceled due to conflicts with database replay (conflicts occur only on the standby node). For details, see <a href="#">STAT_DATABASE_CONFLICTS</a> .
temp_files	numeric	Number of temporary files created by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter <b>log_temp_files</b> .
temp_bytes	numeric	Total amount of data written to temporary files by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter <b>log_temp_files</b> .
deadlocks	bigint	Number of deadlocks detected in this database.
blk_read_time	double precision	Time spent in reading data file blocks by backends in this database (unit: ms).
blk_write_time	double precision	Time spent in writing data file blocks by backends in this database (unit: ms).
stats_reset	timestamp with time zone	Time at which the current statistics were reset.

### 13.2.5.21 GLOBAL\_STAT\_DATABASE

Displays the statistics of each database node in the cluster. (It is used on CNs and the status statistics of each database are not summed up.)

**Table 13-50** GLOBAL\_STAT\_DATABASE columns

Name	Type	Description
node_name	name	Node name.
datid	oid	Database OID.
datname	name	Database name.
numbackends	integer	Number of backends currently connected to this database.
xact_commit	bigint	Number of transactions in this database that have been committed.

Name	Type	Description
xact_rollback	bigint	Number of transactions in this database that have been rolled back.
blks_read	bigint	Number of disk blocks read in this database.
blks_hit	bigint	Number of disk blocks that have been hit in the cache. In this case, data does not need to be read from disks. (The cache includes only the database buffer cache and does not include the file system cache of the OS.)
tup_returned	bigint	Number of live rows fetched by sequential scans and number of index rows returned by index scans in the database.
tup_fetched	bigint	Number of rows returned by the current database through indexes.
tup_inserted	bigint	Number of rows inserted.
tup_updated	bigint	Number of rows updated.
tup_deleted	bigint	Number of rows deleted.
conflicts	bigint	Number of queries canceled due to conflicts with database replay (conflicts occur only on the standby node). For details, see <a href="#">STAT_DATABASE_CONFLICTS</a> .
temp_files	bigint	Number of temporary files created by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter <b>log_temp_files</b> .
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter <b>log_temp_files</b> .
deadlocks	bigint	Number of deadlocks detected in this database.
blk_read_time	double precision	Time spent in reading data file blocks by backends in this database (unit: ms).
blk_write_time	double precision	Time spent in writing data file blocks by backends in this database (unit: ms).

Name	Type	Description
stats_reset	timestamp with time zone	Time at which the current statistics were reset.

### 13.2.5.22 STAT\_DATABASE\_CONFLICTS

Displays statistics about the conflict status of the current database node.

**Table 13-51** STAT\_DATABASE\_CONFLICTS columns

Name	Type	Description
datid	oid	Database ID.
datname	name	Database name.
confl_tablespace	bigint	Number of conflicting tablespaces.
confl_lock	bigint	Number of conflicting locks.
confl_snapshot	bigint	Number of conflicting snapshots.
confl_bufferpin	bigint	Number of conflicting buffers.
confl_deadlock	bigint	Number of conflicting deadlocks.

### 13.2.5.23 SUMMARY\_STAT\_DATABASE\_CONFLICTS

Displays the conflict status statistics of all database nodes in the cluster. It is used on CNs and the conflict status statistics of each node are summed up by database name.

**Table 13-52** SUMMARY\_STAT\_DATABASE\_CONFLICTS columns

Name	Type	Description
datname	name	Database name.
confl_tablespace	bigint	Number of conflicting tablespaces.
confl_lock	bigint	Number of conflicting locks.
confl_snapshot	bigint	Number of conflicting snapshots.
confl_bufferpin	bigint	Number of conflicting buffers.
confl_deadlock	bigint	Number of conflicting deadlocks.

### 13.2.5.24 GLOBAL\_STAT\_DATABASE\_CONFLICTS

Displays statistics about the conflict status of each database node. It is used on CNs and the status information of each database node is not summed up.

**Table 13-53** GLOBAL\_STAT\_DATABASE\_CONFLICTS columns

Name	Type	Description
node_name	name	Node name.
datid	oid	Database ID.
datname	name	Database name.
confl_tablespace	bigint	Number of conflicting tablespaces.
confl_lock	bigint	Number of conflicting locks.
confl_snapshot	bigint	Number of conflicting snapshots.
confl_bufferpin	bigint	Number of conflicting buffers.
confl_deadlock	bigint	Number of conflicting deadlocks.

### 13.2.5.25 STAT\_XACT\_ALL\_TABLES

Displays transaction status information about all ordinary tables and TOAST tables in all schemas on the current node.

**Table 13-54** STAT\_XACT\_ALL\_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetched	bigint	Number of live rows fetched by index scans.

Name	Type	Description
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

### 13.2.5.26 SUMMARY\_STAT\_XACT\_ALL\_TABLES

Displays the transaction status statistics of all ordinary tables and TOAST tables in schemas of all nodes in the cluster. It is used on CNs and the transaction status statistics of tables on each node are summed up by table name.

**Table 13-55** SUMMARY\_STAT\_XACT\_ALL\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	numeric	Number of sequential scans initiated on the table.
seq_tup_read	numeric	Number of live rows fetched by sequential scans.
idx_scan	numeric	Number of index scans initiated on the table.
idx_tup_fetch	numeric	Number of live rows fetched by index scans.
n_tup_ins	numeric	Number of rows inserted.
n_tup_upd	numeric	Number of rows updated.
n_tup_del	numeric	Number of rows deleted.
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

### 13.2.5.27 GLOBAL\_STAT\_XACT\_ALL\_TABLES

Displays the transaction status information of all ordinary tables and TOAST tables in the schemas of each node. (It is used on CNs. The transaction status information of the same table name on different nodes is not summed up.)

**Table 13-56** GLOBAL\_STAT\_XACT\_ALL\_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

### 13.2.5.28 STAT\_XACT\_SYS\_TABLES

Displays transaction status information about the system catalogs in the schemas of the current node.

**Table 13-57** STAT\_XACT\_SYS\_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.

Name	Type	Description
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

### 13.2.5.29 SUMMARY\_STAT\_XACT\_SYS\_TABLES

Displays the transaction status statistics of system catalogs in schemas of all nodes in the cluster. It is used on a CN and the transaction status statistics of tables on each node are summed up by table name.

**Table 13-58** SUMMARY\_STAT\_XACT\_SYS\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	numeric	Number of sequential scans initiated on the table.
seq_tup_read	numeric	Number of live rows fetched by sequential scans.
idx_scan	numeric	Number of index scans initiated on the table.
idx_tup_fetch	numeric	Number of live rows fetched by index scans.
n_tup_ins	numeric	Number of rows inserted.
n_tup_upd	numeric	Number of rows updated.
n_tup_del	numeric	Number of rows deleted.
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

### 13.2.5.30 GLOBAL\_STAT\_XACT\_SYS\_TABLES

Displays the transaction status information of the system catalogs in the schemas of each node. (It is used on CNs. The transaction status information of the same table on different nodes is not summed up.)

**Table 13-59** GLOBAL\_STAT\_XACT\_SYS\_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

### 13.2.5.31 STAT\_XACT\_USER\_TABLES

Displays transaction status information about the user tables in schemas on the current node.

**Table 13-60** STAT\_XACT\_USER\_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.

Name	Type	Description
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose new index columns are not updated).

### 13.2.5.32 SUMMARY\_STAT\_XACT\_USER\_TABLES

Displays the transaction status statistics of user tables in schemas of all nodes in the cluster. It is used on CNs and the transaction status statistics of tables on each node are summed up by table name.

**Table 13-61** SUMMARY\_STAT\_XACT\_USER\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	numeric	Number of sequential scans initiated on the table.
seq_tup_read	numeric	Number of live rows fetched by sequential scans.
idx_scan	numeric	Number of index scans initiated on the table.
idx_tup_fetch	numeric	Number of live rows fetched by index scans.
n_tup_ins	numeric	Number of rows inserted.
n_tup_upd	numeric	Number of rows updated.
n_tup_del	numeric	Number of rows deleted.
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

### 13.2.5.33 GLOBAL\_STAT\_XACT\_USER\_TABLES

Displays the transaction status information of the user tables in the schemas of each node. (It is used on CNs. The transaction status information of the same table on different nodes is not summed up.)

**Table 13-62** GLOBAL\_STAT\_XACT\_USER\_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

### 13.2.5.34 STAT\_XACT\_USER\_FUNCTIONS

Displays statistics about function executions in the current transaction on the current database node.

**Table 13-63** STAT\_XACT\_USER\_FUNCTIONS columns

Name	Type	Description
funcid	oid	Function ID.
schemaname	name	Name of the schema where the function is located.
funcname	name	Function name.
calls	bigint	Number of times the function has been called.
total_time	double precision	Total time spent in this function and all other functions called by it.

Name	Type	Description
self_time	double precision	Time spent in this function, excluding other functions called by it.

### 13.2.5.35 SUMMARY\_STAT\_XACT\_USER\_FUNCTIONS

Displays the summary of function execution statistics in the current transaction on each node in the cluster. (It is used on CNs.)

**Table 13-64** SUMMARY\_STAT\_XACT\_USER\_FUNCTIONS columns

Name	Type	Description
schemaname	name	Name of the schema where the function is located.
funcname	name	Function name.
calls	numeric	Number of times the function has been called.
total_time	double precision	Total time spent in this function and all other functions called by it.
self_time	double precision	Time spent in this function, excluding other functions called by it.

### 13.2.5.36 GLOBAL\_STAT\_XACT\_USER\_FUNCTIONS

Displays statistics about function execution in the current transaction on each node in the cluster. (It is used on CNs.)

**Table 13-65** GLOBAL\_STAT\_XACT\_USER\_FUNCTIONS columns

Name	Type	Description
node_name	name	Node name.
funcid	oid	Function ID.
schemaname	name	Name of the schema where the function is located.
funcname	name	Function name.
calls	bigint	Number of times the function has been called.
total_time	double precision	Total time spent in this function and all other functions called by it.

Name	Type	Description
self_time	double precision	Total time spent in this function, excluding other functions called by it

### 13.2.5.37 STAT\_BAD\_BLOCK

Displays information about table and index read failures on the current node.

**Table 13-66** STAT\_BAD\_BLOCK columns

Name	Type	Description
nodename	text	Node name.
databaseid	integer	Database OID.
tablespaceid	integer	Tablespace OID.
relfilenode	integer	Relationship file node.
bucketid	smallint	ID of the bucket for consistent hashing.
forknum	integer	Fork number.
error_count	integer	Number of errors.
first_time	timestamp with time zone	Time when the page damage occurs for the first time.
last_time	timestamp with time zone	Time when the page damage occurs for the last time.

### 13.2.5.38 SUMMARY\_STAT\_BAD\_BLOCK

Obtains the summary of information about table and index read failures on each node in the cluster. It is used on CNs. The statistics of different DNs is summed up. The value of **first\_time** is the statistics collection start time, and the value of **last\_time** is the statistics collection end time.

**Table 13-67** SUMMARY\_STAT\_BAD\_BLOCK columns

Name	Type	Description
databaseid	integer	Database OID.
tablespaceid	integer	Tablespace OID.
relfilenode	integer	Relationship file node.
forknum	bigint	Fork number.

Name	Type	Description
error_count	bigint	Number of errors.
first_time	timestamp with time zone	Time when the page damage occurs for the first time.
last_time	timestamp with time zone	Time when the page damage occurs for the last time.

### 13.2.5.39 GLOBAL\_STAT\_BAD\_BLOCK

Obtains the information about table and index read failures on each node. It is used on CNs. The read failure information of DN is displayed only, but not summed up.

**Table 13-68** GLOBAL\_STAT\_BAD\_BLOCK columns

Name	Type	Description
node_name	text	Node name
databaseid	integer	OID of the database
tablespaceid	integer	OID of the tablespace
relfilenode	integer	File node of this relation
forknum	integer	Fork number
error_count	integer	Number of errors
first_time	timestamp with time zone	Time when the page damage occurs for the first time.
last_time	timestamp with time zone	Time when the page damage occurs for the last time.

### 13.2.5.40 STAT\_USER\_FUNCTIONS

Displays user-defined function status information in the schemas of the current node. (The language of the function is non-internal language.)

**Table 13-69** STAT\_USER\_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of the function.
schemaname	name	Schema name.
funcname	name	Rename the customized function.

Name	Type	Description
calls	bigint	Number of times the function has been called.
total_time	double precision	Total time spent in this function, including other functions called by it (unit: ms).
self_time	double precision	Time spent in this function, excluding other functions called by it (unit: ms).

### 13.2.5.41 SUMMARY\_STAT\_USER\_FUNCTIONS

Displays the summary of statistics about user-defined functions on each node in the entire cluster. It is used on CNs and the status information of user-defined functions on each node is summed up by function name.

**Table 13-70** SUMMARY\_STAT\_USER\_FUNCTIONS columns

Name	Type	Description
schemaname	name	Schema name.
funcname	name	UDF name.
calls	numeric	Number of times that the function has been called.
total_time	double precision	Total time spent in this function, including other functions called by it (unit: ms).
self_time	double precision	Time spent in this function, excluding other functions called by it (unit: ms).

### 13.2.5.42 GLOBAL\_STAT\_USER\_FUNCTIONS

Displays statistics about user-defined functions on each node in the entire cluster. (It is used on CNs. Status information about functions with the same name on different nodes is not summed up.)

**Table 13-71** GLOBAL\_STAT\_USER\_FUNCTIONS columns

Name	Type	Description
node_name	name	Node name.
funcid	oid	ID of the function.

Name	Type	Description
schemaname	name	Name of the schema where the function is located.
funcname	name	UDF name.
calls	bigint	Number of times the function has been called.
total_time	double precision	Total time spent in this function, including other functions called by it (unit: ms).
self_time	double precision	Time spent in this function, excluding other functions called by it (unit: ms).

## 13.2.6 Workload

### 13.2.6.1 WORKLOAD\_SQL\_COUNT

Displays the distribution of SQL statements in workloads on the current node.

**Table 13-72** WORKLOAD\_SQL\_COUNT columns

Name	Type	Description
workload	name	Workload name.
select_count	bigint	Number of SELECT statements.
update_count	bigint	Number of UPDATE statements.
insert_count	bigint	Number of INSERT statements.
delete_count	bigint	Number of DELETE statements.
ddl_count	bigint	Number of DDL statements.
dml_count	bigint	Number of DML statements.
dcl_count	bigint	Number of DCL statements.

### 13.2.6.2 SUMMARY\_WORKLOAD\_SQL\_COUNT

**SUMMARY\_WORKLOAD\_SQL\_COUNT** displays the distribution of SQL statements in workloads on each CN in the cluster.

**Table 13-73** SUMMARY\_WORKLOAD\_SQL\_COUNT columns

Name	Type	Description
node_name	name	Node name
workload	name	Workload name
select_count	bigint	Number of <b>SELECT</b> statements
update_count	bigint	Number of <b>UPDATE</b> statements
insert_count	bigint	Number of <b>INSERT</b> statements
delete_count	bigint	Number of <b>DELETE</b> statements
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements
dcl_count	bigint	Number of DCL statements

### 13.2.6.3 WORKLOAD\_TRANSACTION

**WORKLOAD\_TRANSACTION** displays the information about transactions on the current node.

**Table 13-74** WORKLOAD\_TRANSACTION columns

Name	Type	Description
workload	name	Workload name
commit_counter	bigint	Number of user transactions committed
rollback_counter	bigint	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: $\mu$ s)
resp_max	bigint	Maximum response time of user transactions (unit: $\mu$ s)
resp_avg	bigint	Average response time of user transactions (unit: $\mu$ s)
resp_total	bigint	Total response time of user transactions (unit: $\mu$ s)
bg_commit_counter	bigint	Number of background transactions committed

Name	Type	Description
bg_rollback_counter	bigint	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: $\mu$ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: $\mu$ s)
bg_resp_avg	bigint	Average response time of background transactions (unit: $\mu$ s)
bg_resp_total	bigint	Total response time of background transactions (unit: $\mu$ s)

### 13.2.6.4 SUMMARY\_WORKLOAD\_TRANSACTION

**SUMMARY\_WORKLOAD\_TRANSACTION** displays the information about transactions in the cluster.

**Table 13-75** SUMMARY\_WORKLOAD\_TRANSACTION columns

Name	Type	Description
workload	name	Workload name
commit_counter	numeric	Number of user transactions committed
rollback_counter	numeric	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: $\mu$ s)
resp_max	bigint	Maximum response time of user transactions (unit: $\mu$ s)
resp_avg	bigint	Average response time of user transactions (unit: $\mu$ s)
resp_total	numeric	Total response time of user transactions (unit: $\mu$ s)
bg_commit_counter	numeric	Number of background transactions committed
bg_rollback_counter	numeric	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: $\mu$ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: $\mu$ s)

Name	Type	Description
bg_resp_avg	bigint	Average response time of background transactions (unit: $\mu$ s)
bg_resp_total	numeric	Total response time of background transactions (unit: $\mu$ s)

### 13.2.6.5 GLOBAL\_WORKLOAD\_TRANSACTION

**GLOBAL\_WORKLOAD\_TRANSACTION** displays the information about workloads on each node.

**Table 13-76** GLOBAL\_WORKLOAD\_TRANSACTION columns

Name	Type	Description
node_name	name	Node name
workload	name	Workload name
commit_counter	bigint	Number of user transactions committed
rollback_counter	bigint	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: $\mu$ s)
resp_max	bigint	Maximum response time of user transactions (unit: $\mu$ s)
resp_avg	bigint	Average response time of user transactions (unit: $\mu$ s)
resp_total	bigint	Total response time of user transactions (unit: $\mu$ s)
bg_commit_counter	bigint	Number of background transactions committed
bg_rollback_counter	bigint	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: $\mu$ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: $\mu$ s)
bg_resp_avg	bigint	Average response time of background transactions (unit: $\mu$ s)
bg_resp_total	bigint	Total response time of background transactions (unit: $\mu$ s)

### 13.2.6.6 WORKLOAD\_SQL\_ELAPSE\_TIME

WORKLOAD\_SQL\_ELAPSE\_TIME collects SELECT, UPDATE, INSERT, and DELETE (SUID) statistics.

**Table 13-77** WORKLOAD\_SQL\_ELAPSE\_TIME columns

Name	Type	Description
workload	name	Workload name.
total_select_elapse	bigint	Total response time of SELECT statements (unit: $\mu$ s).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: $\mu$ s).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: $\mu$ s).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: $\mu$ s).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: $\mu$ s).
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: $\mu$ s).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: $\mu$ s).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: $\mu$ s).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: $\mu$ s).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: $\mu$ s).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: $\mu$ s).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: $\mu$ s).
total_delete_elapse	bigint	Total response time of DELETE statements (unit: $\mu$ s).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: $\mu$ s).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: $\mu$ s).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: $\mu$ s).

### 13.2.6.7 SUMMARY\_WORKLOAD\_SQL\_ELAPSE\_TIME

WORKLOAD\_SQL\_ELAPSE\_TIME collects SELECT, UPDATE, INSERT, and DELETE (SUID) statistics on all CNs.

**Table 13-78** SUMMARY\_WORKLOAD\_SQL\_ELAPSE\_TIME columns

Name	Type	Description
node_name	name	Node name.
workload	name	Workload name.
total_select_elapse	bigint	Total response time of SELECT statements (unit: $\mu$ s).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: $\mu$ s).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: $\mu$ s).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: $\mu$ s).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: $\mu$ s).
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: $\mu$ s).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: $\mu$ s).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: $\mu$ s).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: $\mu$ s).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: $\mu$ s).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: $\mu$ s).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: $\mu$ s).
total_delete_elapse	bigint	Total response time of DELETE statements (unit: $\mu$ s).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: $\mu$ s).

Name	Type	Description
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: $\mu$ s).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: $\mu$ s).

### 13.2.6.8 USER\_TRANSACTION

**USER\_TRANSACTION** collects statistics about transactions executed by users. The monadmin user can view information about transactions executed by all users.

**Table 13-79** USER\_TRANSACTION columns

Name	Type	Description
username	name	Username
commit_counter	bigint	Number of user transactions committed
rollback_counter	bigint	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: $\mu$ s)
resp_max	bigint	Maximum response time of user transactions (unit: $\mu$ s)
resp_avg	bigint	Average response time of user transactions (unit: $\mu$ s)
resp_total	bigint	Total response time of user transactions (unit: $\mu$ s)
bg_commit_counter	bigint	Number of background transactions committed
bg_rollback_counter	bigint	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: $\mu$ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: $\mu$ s)
bg_resp_avg	bigint	Average response time of background transactions (unit: $\mu$ s)
bg_resp_total	bigint	Total response time of background transactions (unit: $\mu$ s)

### 13.2.6.9 GLOBAL\_USER\_TRANSACTION

**GLOBAL\_USER\_TRANSACTION** collects statistics about transactions executed by all users.

**Table 13-80** GLOBAL\_USER\_TRANSACTION columns

Name	Type	Description
node_name	name	Node name
username	name	Username
commit_counter	bigint	Number of user transactions committed
rollback_counter	bigint	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: $\mu$ s)
resp_max	bigint	Maximum response time of user transactions (unit: $\mu$ s)
resp_avg	bigint	Average response time of user transactions (unit: $\mu$ s)
resp_total	bigint	Total response time of user transactions (unit: $\mu$ s)
bg_commit_counter	bigint	Number of background transactions committed
bg_rollback_counter	bigint	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: $\mu$ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: $\mu$ s)
bg_resp_avg	bigint	Average response time of background transactions (unit: $\mu$ s)
bg_resp_total	bigint	Total response time of background transactions (unit: $\mu$ s)

## 13.2.7 Session and Thread

### 13.2.7.1 SESSION\_STAT

**SESSION\_STAT** collects statistics about session status on the current node based on session threads or the **AutoVacuum** thread.

**Table 13-81** SESSION\_STAT columns

Name	Type	Description
sessid	text	Thread start time and ID
statid	integer	Statistics ID
statname	text	Name of the statistics session
statunit	text	Unit of the statistics session
value	bigint	Value of the statistics session

### 13.2.7.2 GLOBAL\_SESSION\_STAT

**GLOBAL\_SESSION\_STAT** collects statistics about session status on each node based on session threads or the **AutoVacuum** thread.

**Table 13-82** GLOBAL\_SESSION\_STAT columns

Name	Type	Description
node_name	name	Node name
sessid	text	Thread start time and ID
statid	integer	Statistics ID
statname	text	Name of the statistics session
statunit	text	Unit of the statistics session
value	bigint	Value of the statistics session

### 13.2.7.3 SESSION\_TIME

**SESSION\_TIME** collects statistics about the running time of session threads and time consumed in each execution phase on the current node.

**Table 13-83** SESSION\_TIME columns

Name	Type	Description
sessid	text	Thread start time and ID
stat_id	integer	Statistics ID
stat_name	text	Session type
value	bigint	Session value

### 13.2.7.4 GLOBAL\_SESSION\_TIME

**GLOBAL\_SESSION\_TIME** collects statistics about the running time of session threads and time consumed in each execution phase on each node.

**Table 13-84** GLOBAL\_SESSION\_TIME columns

Name	Type	Description
node_name	name	Node name
sessid	text	Thread start time and ID
stat_id	integer	Statistics ID
stat_name	text	Session type
value	bigint	Session value

### 13.2.7.5 SESSION\_MEMORY

Collects statistics about memory usage at the session level in the unit of MB, including all the memory allocated to GaussDB and stream threads on DNs for jobs currently executed by users.

**Table 13-85** SESSION\_MEMORY columns

Name	Type	Description
sessid	text	Thread start time and ID.
init_mem	integer	Memory allocated to the currently executed job before the job enters the executor.
used_mem	integer	Memory allocated to the currently executed job.
peak_mem	integer	Peak memory allocated to the currently executed job.

### 13.2.7.6 GLOBAL\_SESSION\_MEMORY

Collects statistics about memory usage at the session level on each node in the unit of MB, including all the memory allocated to GaussDB and stream threads on DNs for jobs currently executed by users.

**Table 13-86** GLOBAL\_SESSION\_MEMORY columns

Name	Type	Description
node_name	name	Node name.
sessid	text	Thread start time and ID.

Name	Type	Description
init_mem	integer	Memory allocated to the currently executed job before the job enters the executor.
used_mem	integer	Memory allocated to the currently executed job.
peak_mem	integer	Peak memory allocated to the currently executed job.

### 13.2.7.7 SESSION\_MEMORY\_DETAIL

**SESSION\_MEMORY\_DETAIL** collects statistics about thread memory usage by MemoryContext node.

**Table 13-87** SESSION\_MEMORY\_DETAIL columns

Name	Type	Description
sessid	text	Thread start time and ID
sesstype	text	Thread name
contextname	text	Name of the memory context
level	smallint	Level of memory context importance
parent	text	Name of the parent memory context
totalsize	bigint	Size of the applied memory (unit: byte)
freesize	bigint	Size of the idle memory (unit: byte)
usedsize	bigint	Size of the used memory (unit: byte)

### 13.2.7.8 GLOBAL\_SESSION\_MEMORY\_DETAIL

**GLOBAL\_SESSION\_MEMORY\_DETAIL** collects statistics about thread memory usage on each node by MemoryContext node.

**Table 13-88** GLOBAL\_SESSION\_MEMORY\_DETAIL columns

Name	Type	Description
node_name	name	Node name
sessid	text	Thread start time and ID
sesstype	text	Thread name
contextname	text	Name of the memory context
level	smallint	Level of memory context importance

Name	Type	Description
parent	text	Name of the parent memory context
totalsize	bigint	Size of the applied memory (unit: byte)
freesize	bigint	Size of the idle memory (unit: byte)
usedsize	bigint	Size of the used memory (unit: byte)

### 13.2.7.9 SESSION\_STAT\_ACTIVITY

Displays information about threads that are running on the current node.

**Table 13-89** SESSION\_STAT\_ACTIVITY columns

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend.
datname	name	Name of the database that the user session connects to in the backend.
pid	bigint	Backend thread ID.
usesysid	oid	OID of the user logging in to the backend.
username	name	Name of the user logging in to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (-1 if a Unix socket is used).
backend_start	timestampwith time zone	Time when this process was started, that is, when the client connected to the server.

Name	Type	Description
xact_start	timestampwith time zone	Time when the current transaction was started (null if no transactions are active). If the current query is the first of its transaction, the value of this column is the same as that of the <b>query_start</b> column.
query_start	timestampwith time zone	Time when the currently active query was started, or time when the last query was started if <b>state</b> is not <b>active</b> . For a stored procedure or function, the first query time is queried and does not change with the running of statements in the stored procedure.
state_change	timestampwith time zone	Time when the state was last changed.
waiting	boolean	Specifies whether the backend is currently waiting for a lock. If the backend is currently waiting for a lock, the value is <b>true</b> .



Name	Type	Description
unique_sql_id	bigint	Unique SQL statement ID.
trace_id	text	Driver-specific trace ID, which is associated with an application request.

### 13.2.7.10 GLOBAL\_SESSION\_STAT\_ACTIVITY

Displays information about threads that are running on each node in the cluster.

**Table 13-90** GLOBAL\_SESSION\_STAT\_ACTIVITY columns

Name	Type	Description
coorname	text	CN name.
datid	oid	OID of the database that the user session connects to in the backend.
datname	text	Name of the database that the user session connects to in the backend.
pid	bigint	Backend thread ID.
usesysid	oid	OID of the user logging in to the backend.
username	text	Name of the user logged in to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of <b>client_addr</b> . This column will be non-null only for IP connections and only when <b>log_hostname</b> is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (-1 if a Unix socket is used).
backend_start	timestampwith time zone	Time when this process was started, that is, when the client connected to the server.

Name	Type	Description
xact_start	timestampwith time zone	Time when the current transaction was started ( <b>null</b> if no transactions are active). If the current query is the first of its transaction, the value of this column is the same as that of the <b>query_start</b> column.
query_start	timestampwith time zone	Time when the currently active query was started, or time when the last query was started if <b>state</b> is not <b>active</b> . For a stored procedure or function, the first query time is queried and does not change with the running of statements in the stored procedure.
state_change	timestampwith time zone	Time when the state was last changed.
waiting	Boolean	Specifies whether the backend is currently waiting for a lock. If the backend is currently waiting for a lock, the value is <b>true</b> . If a lock is being waited for on a DN, determine the DN based on the <b>wait_status</b> column in the PG_THREAD_WAIT_STATUS view and query the <b>waiting</b> column in PG_STAT_ACTIVITY on the DN to check whether a lock wait occurs.

Name	Type	Description
state	text	<p>Overall status of this backend. Its value can be:</p> <ul style="list-style-type: none"> <li>• <b>active</b>: The backend is executing a query.</li> <li>• <b>idle</b>: The backend is waiting for a new client command.</li> <li>• <b>idle in transaction</b>: The backend is in a transaction, but is not currently executing a query.</li> <li>• <b>idle in transaction (aborted)</b>: The backend is in a transaction, but there are statements failed in the transaction.</li> <li>• <b>fastpath function call</b>: The backend is executing a fast-path function.</li> <li>• <b>disabled</b>: This state is reported if <b>track_activities</b> is disabled in this backend.</li> </ul> <p><b>NOTE</b> Common users can view their own session status only. The state information of other accounts is empty. For example, after the <b>judy</b> user is connected to the database, the state information of the <b>joe</b> user and the initial user <b>omm</b> in pg_stat_activity is empty.</p> <pre>gaussdb=# SELECT datname, username, usesysid,state,pid FROM pg_stat_activity; datname   username   usesysid   state   pid -----+-----+-----+-----+----- +-----+-----+-----+-----+----- testdb   omm   10     139968752121616 testdb   omm   10     139968903116560 db_tpcds   judy   16398   active   139968391403280 testdb   omm   10     139968643069712 testdb   omm   10     139968680818448 testdb   joe   16390     139968563377936 (6 rows)</pre>
resource_pool	name	Resource pool used by the user.
query_id	bigint	ID of a query.
query	text	Text of this backend's latest query. If the value of <b>state</b> is <b>active</b> , this column shows the ongoing query. In all other states, it shows the last query that was executed.
unique_sql_id	bigint	Unique SQL statement ID.

Name	Type	Description
trace_id	text	Driver-specific trace ID, which is associated with an application request.

### 13.2.7.11 THREAD\_WAIT\_STATUS

In this view, you can check the blocking and waiting status of the backend threads and auxiliary threads on the current node. For detailed events, see [Table 13-175](#).

**Table 13-91** THREAD\_WAIT\_STATUS columns

Name	Type	Description
node_name	text	Current node name.
db_name	text	Database name.
thread_name	text	Thread name.
query_id	bigint	Query ID. The value of this column is the same as that of <b>debug_query_id</b> .
tid	bigint	Thread ID of the current thread.
sessionid	bigint	Session ID.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.
tlevel	integer	Level of the streaming thread.
smpid	integer	Concurrent thread ID.
wait_status	text	Waiting status of the current thread. For details about the waiting status, see <a href="#">Table 13-175</a> .
wait_event	text	If <b>wait_status</b> is <b>acquire lock</b> , <b>acquire lwlock</b> , or <b>wait io</b> , this column describes the lock, lightweight lock, or I/O information. Otherwise, this column is empty.
locktag	text	Information about the lock that the current thread is waiting for.
lockmode	text	Lock mode that the current thread is waiting to obtain. The values include table-level lock, row-level lock, and page-level lock modes.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock.

Name	Type	Description
global_sessionid	text	Global session ID.

### 13.2.7.12 GLOBAL\_THREAD\_WAIT\_STATUS

In this view, you can check the blocking and waiting status of backend threads and auxiliary threads on all nodes. For details about the events, see [Table 13-175](#).

In GLOBAL\_THREAD\_WAIT\_STATUS, you can see all the call hierarchy relationships between threads of the SQL statements on all nodes in the cluster, and the blocking status for each thread. With this view, you can easily locate the causes of process hang and similar issues.

The definitions of GLOBAL\_THREAD\_WAIT\_STATUS and THREAD\_WAIT\_STATUS are the same, because the essence of the GLOBAL\_THREAD\_WAIT\_STATUS view is the query summary of the THREAD\_WAIT\_STATUS view on each node in the cluster.

**Table 13-92** GLOBAL\_THREAD\_WAIT\_STATUS columns

Name	Type	Description
node_name	text	Node name.
db_name	text	Database name.
thread_name	text	Thread name.
query_id	bigint	Query ID. The value of this column is the same as that of <b>debug_query_id</b> .
tid	bigint	Thread ID of the current thread.
sessionid	bigint	Session ID.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.
tlevel	integer	Level of the streaming thread.
smpid	integer	Concurrent thread ID.
wait_status	text	Waiting status of the current thread. For details about the waiting status, see <a href="#">Table 13-175</a> .
wait_event	text	If <b>wait_status</b> is <b>acquire lock</b> , <b>acquire lwlock</b> , or <b>wait io</b> , this column describes the lock, lightweight lock, or I/O information. Otherwise, this column is empty.

Name	Type	Description
locktag	text	Information about the lock that the current thread is waiting for.
lockmode	text	Lock mode that the current thread is waiting to obtain. The values include table-level lock, row-level lock, and page-level lock modes.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock.
global_sessionid	text	Global session ID.

### 13.2.7.13 LOCAL\_THREADPOOL\_STATUS

LOCAL\_THREADPOOL\_STATUS displays the status of worker threads and sessions in a thread pool. This view is valid only when **enable\_thread\_pool** is set to **on**.

**Table 13-93** LOCAL\_THREADPOOL\_STATUS columns

Name	Type	Description
node_name	text	Node name.
group_id	integer	ID of the thread pool group.
bind_numa_id	integer	NUMA ID to which the thread pool group is bound.
bind_cpu_number	integer	Information about the CPU to which the thread pool group is bound. If no CPUs are bound, the value is <b>NULL</b> .
listener	integer	Number of listener threads in the thread pool group.

Name	Type	Description
worker_info	text	Information about threads in the thread pool, including: <ul style="list-style-type: none"> <li>● <b>default</b>: Number of initial threads in the thread pool group</li> <li>● <b>new</b>: Number of new threads in the thread pool group</li> <li>● <b>expect</b>: Expected number of threads in the thread pool group</li> <li>● <b>actual</b>: Actual number of threads in the thread pool group</li> <li>● <b>idle</b>: Number of idle threads in the thread pool group</li> <li>● <b>pending</b>: Number of pending threads in the thread pool group</li> </ul>
session_info	text	Information about sessions in the thread pool, including: <ul style="list-style-type: none"> <li>● <b>total</b>: Total number of sessions in the thread pool group.</li> <li>● <b>waiting</b>: Number of sessions pending scheduling in the thread pool group.</li> <li>● <b>running</b>: Number of running sessions in the thread pool group.</li> <li>● <b>idle</b>: Number of idle sessions in the thread pool group.</li> </ul>
stream_info	text	Information about streams in the thread pool, including: <ul style="list-style-type: none"> <li>● <b>total</b>: total number of stream threads in the thread pool group.</li> <li>● <b>running</b>: number of running stream threads in the thread pool group.</li> <li>● <b>idle</b>: number of idle stream threads in the thread pool group.</li> </ul>

### 13.2.7.14 GLOBAL\_THREADPOOL\_STATUS

**GLOBAL\_THREADPOOL\_STATUS** displays the status of worker threads and sessions in thread pools on all nodes. Columns in this view are the same as those in [Table 13-93](#).

### 13.2.7.15 SESSION\_CPU\_RUNTIME

**SESSION\_CPU\_RUNTIME** displays information about CPU usage of ongoing complex jobs executed by the current user.

**Table 13-94** SESSION\_CPU\_RUNTIME columns

Name	Type	Description
datid	oid	OID of the database that this backend is connected to.
username	name	Name of the user logged in to the backend.
pid	bigint	Backend thread ID.
start_time	timestamp with time zone	Time when the statement execution starts. For a stored procedure or function, the first query time is queried and does not change with the running of statements in the stored procedure.
min_cpu_time	bigint	Minimum CPU time of the statement across all DNs (unit: ms).
max_cpu_time	bigint	Maximum CPU time of the statement across all DNs (unit: ms).
total_cpu_time	bigint	Total CPU time of the statement across all DNs (unit: ms).
query	text	Statement being executed.
top_cpu_dn	text	Top <i>N</i> CPU usage.

### 13.2.7.16 SESSION\_MEMORY\_RUNTIME

SESSION\_MEMORY\_RUNTIME displays information about memory usage of ongoing complex jobs executed by the current user.

**Table 13-95** SESSION\_MEMORY\_RUNTIME columns

Name	Type	Description
datid	oid	OID of the database that this backend is connected to.
username	name	Name of the user logged in to the backend.
pid	bigint	Backend thread ID.
start_time	timestamp with time zone	Time when the statement execution starts. For a stored procedure or function, the first query time is queried and does not change with the running of statements in the stored procedure.

Name	Type	Description
min_peak_memory	integer	Minimum peak memory of the statement among all DNs (unit: MB).
max_peak_memory	integer	Maximum peak memory of the statement among all DNs (unit: MB).
spill_info	text	Statement spill information on all DNs. <ul style="list-style-type: none"> <li>• <b>None:</b> No data is spilled to disks on all DNs.</li> <li>• <b>All:</b> Data is spilled to disks on all DNs.</li> <li>• <b>[a:b]:</b> The statement has been spilled to disks on <i>a</i> of <i>b</i> DNs.</li> </ul>
query	text	Statement being executed.
top_mem_dn	text	Top N memory usage.

### 13.2.7.17 LOCAL\_ACTIVE\_SESSION

LOCAL\_ACTIVE\_SESSION displays samples in the **ACTIVE SESSION PROFILE** memory on the current node.

**Table 13-96** LOCAL\_ACTIVE\_SESSION columns

Name	Type	Description
sampleid	bigint	Sample ID.
sample_time	timestamp with time zone	Sampling time.
need_flush_sample	Boolean	Specifies whether the sample needs to be flushed to disks.
databaseid	oid	Database ID.
thread_id	bigint	Thread ID.
sessionid	bigint	Session ID.
start_time	timestamp with time zone	Start time of a session.
event	text	Event name.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.

Name	Type	Description
tlevel	integer	Level of the streaming thread. The value corresponds to the level (ID) of the execution plan.
smpid	integer	Concurrent thread ID in SMP execution mode.
userid	oid	ID of a session user.
application_name	text	Name of an application.
client_addr	inet	IP address of a client.
client_hostname	text	Name of a client.
client_port	integer	TCP port number used by a client to communicate with the backend.
query_id	bigint	Debug query ID.
unique_query_id	bigint	Unique query ID.
user_id	oid	User ID in the key of the unique query.
cn_id	integer	A CN ID on a DN indicates that the unique SQL statement comes from a CN ID in a key of the unique query on a CN.
unique_query	text	Standardized text string of the unique SQL statement.
locktag	text	Information of a lock that the session waits for, which can be parsed using <b>locktag_decode</b> .
lockmode	text	Mode of a lock that the session waits for.
block_sessionid	bigint	Blocks a session from obtaining the session ID of a lock if the session is waiting for the lock.
final_block_sessionid	bigint	ID of the blocked session at the source end.
wait_status	text	Provides more details about an event column.
global_sessionid	text	Global session ID.
xact_start_time	timestamp with time zone	Start time of the transaction.

Name	Type	Description
query_start_time	timestamp with time zone	Time when the statement starts to be executed.
state	text	Current statement state. The value can be <b>active</b> , <b>idle in transaction</b> , <b>fastpath function call</b> , <b>idle in transaction (aborted)</b> , <b>disabled</b> , or <b>retrying</b> .

### 13.2.7.18 GLOBAL\_ACTIVE\_SESSION

GLOBAL\_ACTIVE\_SESSION displays a summary of samples in the **ACTIVE SESSION PROFILE** memory on all nodes.

**Table 13-97** GLOBAL\_ACTIVE\_SESSION columns

Name	Type	Description
node_name	text	Node name.
sampleid	bigint	Sample ID.
sample_time	timestamp without time zone	Sampling time.
need_flush_sample	Boolean	Specifies whether the sample needs to be flushed to disks.
databaseid	oid	Database ID.
thread_id	bigint	Thread ID.
sessionid	bigint	Session ID.
start_time	timestamp without time zone	Start time of a session.
event	text	Event name.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.
tlevel	integer	Level of the streaming thread. The value corresponds to the level (ID) of the execution plan.
smpid	integer	Concurrent thread ID in SMP execution mode.
userid	oid	ID of a session user.

Name	Type	Description
application_name	text	Name of an application.
client_addr	inet	IP address of a client.
client_hostname	text	Name of a client.
client_port	integer	TCP port number used by a client to communicate with the backend.
query_id	bigint	Debug query ID.
unique_query_id	bigint	Unique query ID.
user_id	oid	User ID in the key of the unique query.
cn_id	integer	A CN ID on a DN indicates that the unique SQL statement comes from a CN ID in a key of the unique query on a CN.
unique_query	text	Standardized text string of the unique SQL statement.
locktag	text	Information of a lock that the session waits for, which can be parsed using <b>locktag_decode</b> .
lockmode	text	Mode of a lock that the session waits for.
block_sessionid	bigint	Blocks a session from obtaining the session ID of a lock if the session is waiting for the lock.
final_block_sessionid	bigint	ID of the blocked session at the source end.
wait_status	text	Provides more details about an event column.
global_sessionid	text	Global session ID.
xact_start_time	timestamp with time zone	Start time of the transaction.
query_start_time	timestamp with time zone	Time when the statement starts to be executed.
state	text	Current statement state. The value can be <b>active</b> , <b>idle in transaction</b> , <b>fastpath function call</b> , <b>idle in transaction (aborted)</b> , <b>disabled</b> , or <b>retrying</b> .

## 13.2.8 Transaction

### 13.2.8.1 TRANSACTIONS\_RUNNING\_XACTS

Displays information about running transactions on the current node.

**Table 13-98** TRANSACTIONS\_RUNNING\_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM.
gxid	xid	Transaction ID.
state	tinyint	Transaction status ( <b>3</b> : prepared; <b>0</b> : starting).
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	Boolean	Specifies whether the current transaction is lazy vacuum.
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the <b>prepared</b> state. If the state is not <b>prepared</b> , the value is <b>0</b> .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Minimum CSN of a local active transaction.

### 13.2.8.2 SUMMARY\_TRANSACTIONS\_RUNNING\_XACTS

Displays information about the running transactions on each CN in the cluster. The column content is the same as that of TRANSACTIONS\_RUNNING\_XACTS.

**Table 13-99** SUMMARY\_TRANSACTIONS\_RUNNING\_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM.
gxid	xid	Transaction ID.
state	tinyint	Transaction status ( <b>3</b> : prepared; <b>0</b> : starting).
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.

Name	Type	Description
vacuum	Boolean	Specifies whether the current transaction is lazy vacuum.
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the <b>prepared</b> state. If the state is not <b>prepared</b> , the value is <b>0</b> .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Minimum CSN of a local active transaction.

### 13.2.8.3 GLOBAL\_TRANSACTIONS\_RUNNING\_XACTS

Displays information about the running transactions on each node in the cluster.

**Table 13-100** GLOBAL\_TRANSACTIONS\_RUNNING\_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM.
gxid	xid	Transaction ID.
state	tinyint	Transaction status ( <b>3</b> : prepared; <b>0</b> : starting).
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	Boolean	Specifies whether the current transaction is lazy vacuum.
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the <b>prepared</b> state. If the state is not <b>prepared</b> , the value is <b>0</b> .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Minimum CSN of a local active transaction.

### 13.2.8.4 TRANSACTIONS\_PREPARED\_XACTS

**TRANSACTIONS\_PREPARED\_XACTS** displays information about transactions that are currently prepared for two-phase commit.

**Table 13-101** TRANSACTIONS\_PREPARED\_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user that executes the transaction
database	name	Name of the database in which the transaction is executed

### 13.2.8.5 SUMMARY\_TRANSACTIONS\_PREPARED\_XACTS

**SUMMARY\_TRANSACTIONS\_PREPARED\_XACTS** displays information about the transactions that are ready for two-phase commit on each CN in the cluster.

**Table 13-102** SUMMARY\_TRANSACTIONS\_PREPARED\_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user that executes the transaction
database	name	Name of the database in which the transaction is executed

### 13.2.8.6 GLOBAL\_TRANSACTIONS\_PREPARED\_XACTS

**GLOBAL\_TRANSACTIONS\_PREPARED\_XACTS** displays information about the transactions that are currently prepared for two-phase commit on each node.

**Table 13-103** GLOBAL\_TRANSACTIONS\_PREPARED\_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user that executes the transaction
database	name	Name of the database in which the transaction is executed

## 13.2.9 Query

### 13.2.9.1 STATEMENT

Obtains information about executed statements (normalized SQL statements) on the current node. You can view all statistics about normalized SQL statements received by the CN, whereas you can view only the statistics about normalized SQL statements executed on the current DN.

#### NOTE

- **unique\_sql\_id** generated by different **savepoint\_name** values is different. When a large number of **savepoint\_name** is used, the number of **unique\_sql\_id** values generated in the system increases rapidly. If the number of **unique\_sql\_id** values is greater than the number of **instr\_unique\_sql\_count** values, the newly generated **unique\_sql\_id** information is not counted.
- In the current version, the keyword FOR UPDATE cannot be identified or normalized. For example, **SELECT \* FROM table;** and **SELECT \* FROM table FOR UPDATE WAIT N;** are normalized into the same normalized SQL statement and displayed in the **query** column.

**Table 13-104** STATEMENT columns

Name	Type	Description
node_name	name	Node name.
node_id	integer	Node ID ( <b>node_id</b> in <b>pgxc_node</b> ).
user_name	name	Username.
user_id	oid	OID of the user.
unique_sql_id	bigint	ID of the normalized SQL statement.

Name	Type	Description
query	text	Normalized SQL statement. Note: The length is controlled by <b>track_activity_query_size</b> .
n_calls	bigint	Number of calls.
min_elapse_time	bigint	Minimum execution time of the SQL statement in the kernel (unit: $\mu$ s).
max_elapse_time	bigint	Maximum execution time of the SQL statement in the kernel (unit: $\mu$ s).
total_elapse_time	bigint	Total execution time of the SQL statement in the kernel (unit: $\mu$ s).
n_returned_rows	bigint	Number of rows in the result set returned by the SELECT statement.
n_tuples_fetched	bigint	Number of rows randomly scanned.
n_tuples_returned	bigint	Number of rows sequentially scanned.
n_tuples_inserted	bigint	Number of rows inserted.
n_tuples_updated	bigint	Number of rows updated.
n_tuples_deleted	bigint	Number of rows deleted.
n_blocks_fetched	bigint	Number of buffer block access times.
n_blocks_hit	bigint	Number of buffer block hits.
n_soft_parse	bigint	Number of soft parsing times. The value of <b>n_soft_parse</b> plus the value of <b>n_hard_parse</b> may be greater than the value of <b>n_calls</b> because the number of subqueries is not counted in the value of <b>n_calls</b> .
n_hard_parse	bigint	Number of hard parsing times. The value of <b>n_soft_parse</b> plus the value of <b>n_hard_parse</b> may be greater than the value of <b>n_calls</b> because the number of subqueries is not counted in the value of <b>n_calls</b> .
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: $\mu$ s).
cpu_time	bigint	CPU time (unit: $\mu$ s).
execution_time	bigint	Execution time in the executor (unit: $\mu$ s).
parse_time	bigint	SQL parsing time (unit: $\mu$ s).
plan_time	bigint	SQL plan generation time (unit: $\mu$ s).

Name	Type	Description
rewrite_time	bigint	SQL rewriting time (unit: $\mu$ s).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: $\mu$ s).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: $\mu$ s).
data_io_time	bigint	I/O time (unit: $\mu$ s).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, CNs communicate with each other, CNs communicate with customer service ends, and CNs communicate with DN through physical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <code>{"time":xxx, "n_calls":xxx, "size":xxx}</code>
net_rcv_info	text	Network status of messages received through a physical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, CNs communicate with each other, CNs communicate with customer service ends, and CNs communicate with DN through physical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <code>{"time":xxx, "n_calls":xxx, "size":xxx}</code>
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, DN of different shards communicate with each other through logical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <code>{"time":xxx, "n_calls":xxx, "size":xxx}</code>

Name	Type	Description
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, DNs of different shards communicate with each other through logical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <code>{"time":xxx, "n_calls":xxx, "size":xxx}</code>
last_updated	timestamp with time zone	Last time when the statement was updated.
sort_count	bigint	Sorting count.
sort_time	bigint	Sorting duration (unit: $\mu$ s).
sort_mem_used	bigint	Size of work memory used during sorting (unit: KB).
sort_spill_count	bigint	Count of file writing when data is flushed to disks during sorting.
sort_spill_size	bigint	File size used when data is flushed to disks during sorting (unit: KB).
hash_count	bigint	Hashing count.
hash_time	bigint	Hashing duration (unit: $\mu$ s).
hash_mem_used	bigint	Size of work memory used during hashing (unit: KB).
hash_spill_count	bigint	Number of file writes when data is flushed to disks during hashing.
hash_spill_size	bigint	File size used when data is flushed to disks during hashing (unit: KB).
parent_unique_sql_id	bigint	Unique ID of the parent SQL statement. The value is <b>0</b> for a non-stored procedure substatement.

 **NOTE**

**n\_calls** indicates the actual number of calling times. For the FETCH statement in the stored procedure, the actual number of triggering times of the FETCH statement is the increase times of **n\_calls** of the statement actually executed by the cursor.

### 13.2.9.2 SUMMARY\_STATEMENT

Displays all information (including DNs) about executed statements (normalized SQL statements) on each CN.

 NOTE

In the current version, the keyword FOR UPDATE cannot be identified or normalized. For example, **SELECT \* FROM table;** and **SELECT \* FROM table FOR UPDATE WAIT N;** are normalized into the same normalized SQL statement and displayed in the **query** column.

**Table 13-105** SUMMARY\_STATEMENT columns

Name	Type	Description
node_name	name	Node name.
node_id	integer	Node ID ( <b>node_id</b> in <b>pgxc_node</b> ).
user_name	name	Username.
user_id	oid	OID of the user.
unique_sql_id	bigint	ID of the normalized SQL statement.
query	text	Normalized SQL statement. Note: The length is controlled by <b>track_activity_query_size</b> .
n_calls	bigint	Number of calls.
min_elapse_time	bigint	Minimum execution time of the SQL statement in the kernel (unit: $\mu$ s).
max_elapse_time	bigint	Maximum execution time of the SQL statement in the kernel (unit: $\mu$ s).
total_elapse_time	bigint	Total execution time of the SQL statement in the kernel (unit: $\mu$ s).
n_returned_rows	bigint	Number of rows in the result set returned by the SELECT statement.
n_tuples_fetched	bigint	Number of rows randomly scanned.
n_tuples_returned	bigint	Number of rows sequentially scanned.
n_tuples_inserted	bigint	Number of rows inserted.
n_tuples_updated	bigint	Number of rows updated.
n_tuples_deleted	bigint	Number of rows deleted.
n_blocks_fetched	bigint	Number of buffer block access times.
n_blocks_hit	bigint	Number of buffer block hits.
n_soft_parse	bigint	Number of soft parsing times.

Name	Type	Description
n_hard_parse	bigint	Number of hard parsing times.
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: $\mu$ s).
cpu_time	bigint	CPU time (unit: $\mu$ s).
execution_time	bigint	Execution time in the executor (unit: $\mu$ s).
parse_time	bigint	SQL parsing time (unit: $\mu$ s).
plan_time	bigint	SQL plan generation time (unit: $\mu$ s).
rewrite_time	bigint	SQL rewriting time (unit: $\mu$ s).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: $\mu$ s).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: $\mu$ s).
data_io_time	bigint	I/O time (unit: $\mu$ s).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, CNs communicate with each other, CNs communicate with customer service ends, and CNs communicate with DN through physical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}
net_rcv_info	text	Network status of messages received through a physical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, CNs communicate with each other, CNs communicate with customer service ends, and CNs communicate with DN through physical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}

Name	Type	Description
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, DNs of different shards communicate with each other through logical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}
net_stream_rcv_info	text	Network status of messages received through a logical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, DNs of different shards communicate with each other through logical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}
last_updated	timestamp with time zone	Last time when the statement was updated.
sort_count	bigint	Sorting count.
sort_time	bigint	Sorting duration (unit: $\mu$ s).
sort_mem_used	bigint	Size of work memory used during sorting (unit: KB).
sort_spill_count	bigint	Count of file writing when data is flushed to disks during sorting.
sort_spill_size	bigint	File size used when data is flushed to disks during sorting (unit: KB).
hash_count	bigint	Hashing count.
hash_time	bigint	Hashing duration (unit: $\mu$ s).
hash_mem_used	bigint	Size of work memory used during hashing (unit: KB).
hash_spill_count	bigint	Number of file writes when data is flushed to disks during hashing.
hash_spill_size	bigint	File size used when data is flushed to disks during hashing (unit: KB).

Name	Type	Description
parent_unique_sql_id	bigint	Unique ID of the parent SQL statement. The value is 0 for a non-stored procedure substatement.

### 13.2.9.3 STATEMENT\_COUNT

Displays statistics about five types of running statements (SELECT, INSERT, UPDATE, DELETE, and MERGE INTO) as well as DDL, DML, and DCL statements on the current node of the database.

#### NOTE

An administrator can query the STATEMENT\_COUNT view to view the statistics of all users on the current node. When the cluster or node is restarted, the statistics are cleared and the counting restarts. The system counts when a node receives a query, including a query inside the cluster. For example, when a CN receives a query and distributes multiple queries to DN, the queries are counted accordingly on the DNs.

**Table 13-106** STATEMENT\_COUNT columns

Name	Type	Description
node_name	text	Node name
user_name	text	Username
select_count	bigint	Statistical result of the <b>SELECT</b> statement
update_count	bigint	Statistical result of the <b>UPDATE</b> statement
insert_count	bigint	Statistical result of the <b>INSERT</b> statement
delete_count	bigint	Statistical result of the <b>DELETE</b> statement
mergeinto_count	bigint	Statistical result of the <b>MERGE INTO</b> statement
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements
dcl_count	bigint	Number of DCL statements
total_select_elapse	bigint	Total response time of <b>SELECT</b> statements (unit: $\mu$ s)
avg_select_elapse	bigint	Average response time of <b>SELECT</b> statements (unit: $\mu$ s)
max_select_elapse	bigint	Maximum response time of <b>SELECT</b> statements (unit: $\mu$ s)
min_select_elapse	bigint	Minimum response time of <b>SELECT</b> statements (unit: $\mu$ s)

Name	Type	Description
total_update_elapse	bigint	Total response time of <b>UPDATE</b> statements (unit: $\mu$ s)
avg_update_elapse	bigint	Average response time of <b>UPDATE</b> statements (unit: $\mu$ s)
max_update_elapse	bigint	Maximum response time of <b>UPDATE</b> statements (unit: $\mu$ s)
min_update_elapse	bigint	Minimum response time of <b>UPDATE</b> statements (unit: $\mu$ s)
total_insert_elapse	bigint	Total response time of <b>INSERT</b> statements (unit: $\mu$ s)
avg_insert_elapse	bigint	Average response time of <b>INSERT</b> statements (unit: $\mu$ s)
max_insert_elapse	bigint	Maximum response time of <b>INSERT</b> statements (unit: $\mu$ s)
min_insert_elapse	bigint	Minimum response time of <b>INSERT</b> statements (unit: $\mu$ s)
total_delete_elapse	bigint	Total response time of <b>DELETE</b> statements (unit: $\mu$ s)
avg_delete_elapse	bigint	Average response time of <b>DELETE</b> statements (unit: $\mu$ s)
max_delete_elapse	bigint	Maximum response time of <b>DELETE</b> statements (unit: $\mu$ s)
min_delete_elapse	bigint	Minimum response time of <b>DELETE</b> statements (unit: $\mu$ s)

#### 13.2.9.4 GLOBAL\_STATEMENT\_COUNT

**GLOBAL\_STATEMENT\_COUNT** displays statistics about four types of running statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) as well as DDL, DML, and DCL statements on each node of the database.

**Table 13-107** GLOBAL\_STATEMENT\_COUNT columns

Name	Type	Description
node_name	text	Node name
user_name	text	Username
select_count	bigint	Statistical result of the <b>SELECT</b> statement

Name	Type	Description
update_count	bigint	Statistical result of the <b>UPDATE</b> statement
insert_count	bigint	Statistical result of the <b>INSERT</b> statement
delete_count	bigint	Statistical result of the <b>DELETE</b> statement
mergeinto_count	bigint	Statistical result of the <b>MERGE INTO</b> statement
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements
dcl_count	bigint	Number of DCL statements
total_select_elapse	bigint	Total response time of <b>SELECT</b> statements (unit: $\mu$ s)
avg_select_elapse	bigint	Average response time of <b>SELECT</b> statements (unit: $\mu$ s)
max_select_elapse	bigint	Maximum response time of <b>SELECT</b> statements (unit: $\mu$ s)
min_select_elapse	bigint	Minimum response time of <b>SELECT</b> statements (unit: $\mu$ s)
total_update_elapse	bigint	Total response time of <b>UPDATE</b> statements (unit: $\mu$ s)
avg_update_elapse	bigint	Average response time of <b>UPDATE</b> statements (unit: $\mu$ s)
max_update_elapse	bigint	Maximum response time of <b>UPDATE</b> statements (unit: $\mu$ s)
min_update_elapse	bigint	Minimum response time of <b>UPDATE</b> statements (unit: $\mu$ s)
total_insert_elapse	bigint	Total response time of <b>INSERT</b> statements (unit: $\mu$ s)
avg_insert_elapse	bigint	Average response time of <b>INSERT</b> statements (unit: $\mu$ s)
max_insert_elapse	bigint	Maximum response time of <b>INSERT</b> statements (unit: $\mu$ s)
min_insert_elapse	bigint	Minimum response time of <b>INSERT</b> statements (unit: $\mu$ s)
total_delete_elapse	bigint	Total response time of <b>DELETE</b> statements (unit: $\mu$ s)
avg_delete_elapse	bigint	Average response time of <b>DELETE</b> statements (unit: $\mu$ s)

Name	Type	Description
max_delete_elapse	bigint	Maximum response time of <b>DELETE</b> statements (unit: $\mu$ s)
min_delete_elapse	bigint	Minimum response time of <b>DELETE</b> statements (unit: $\mu$ s)

### 13.2.9.5 SUMMARY\_STATEMENT\_COUNT

**SUMMARY\_STATEMENT\_COUNT** displays statistics about four types of running statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) as well as DDL, DML, and DCL statements on all nodes (CNs and DN) of the database.

**Table 13-108** SUMMARY\_STATEMENT\_COUNT columns

Name	Type	Description
user_name	text	Username
select_count	numeric	Statistical result of the <b>SELECT</b> statement
update_count	numeric	Statistical result of the <b>UPDATE</b> statement
insert_count	numeric	Statistical result of the <b>INSERT</b> statement
delete_count	numeric	Statistical result of the <b>DELETE</b> statement
mergeinto_count	numeric	Statistical result of the <b>MERGE INTO</b> statement
ddl_count	numeric	Number of DDL statements
dml_count	numeric	Number of DML statements
dcl_count	numeric	Number of DCL statements
total_select_elapse	numeric	Total response time of <b>SELECT</b> statements (unit: $\mu$ s)
avg_select_elapse	bigint	Average response time of <b>SELECT</b> statements (unit: $\mu$ s)
max_select_elapse	bigint	Maximum response time of <b>SELECT</b> statements (unit: $\mu$ s)
min_select_elapse	bigint	Minimum response time of <b>SELECT</b> statements (unit: $\mu$ s)
total_update_elapse	numeric	Total response time of <b>UPDATE</b> statements (unit: $\mu$ s)

Name	Type	Description
avg_update_elapsed	bigint	Average response time of <b>UPDATE</b> statements (unit: $\mu$ s)
max_update_elapsed	bigint	Maximum response time of <b>UPDATE</b> statements (unit: $\mu$ s)
min_update_elapsed	bigint	Minimum response time of <b>UPDATE</b> statements (unit: $\mu$ s)
total_insert_elapsed	numeric	Total response time of <b>INSERT</b> statements (unit: $\mu$ s)
avg_insert_elapsed	bigint	Average response time of <b>INSERT</b> statements (unit: $\mu$ s)
max_insert_elapsed	bigint	Maximum response time of <b>INSERT</b> statements (unit: $\mu$ s)
min_insert_elapsed	bigint	Minimum response time of <b>INSERT</b> statements (unit: $\mu$ s)
total_delete_elapsed	numeric	Total response time of <b>DELETE</b> statements (unit: $\mu$ s)
avg_delete_elapsed	bigint	Average response time of <b>DELETE</b> statements (unit: $\mu$ s)
max_delete_elapsed	bigint	Maximum response time of <b>DELETE</b> statements (unit: $\mu$ s)
min_delete_elapsed	bigint	Minimum response time of <b>DELETE</b> statements (unit: $\mu$ s)

### 13.2.9.6 STATEMENT\_HISTORY

Displays information about statements executed on the current node. The result can be queried only in the system database but cannot be queried in the user database.

 **NOTE**

In the current version, the keyword FOR UPDATE cannot be identified or normalized. For example, **SELECT \* FROM table;** and **SELECT \* FROM table FOR UPDATE WAIT N;** are normalized into the same normalized SQL statement and displayed in the **query** column. SQL statements containing the keyword FOR UPDATE can be distinguished by the **query\_plan** column. The execution plan contains '**lockRows**'.

**Table 13-109** STATEMENT\_HISTORY columns

Name	Type	Description
dbname	name	Database name.

Name	Type	Description
schemaname	name	Schema name.
origin_node	integer	Node name.
user_name	name	Username.
application_name	text	Name of the application that sends a request.
client_addr	text	IP address of the client that sends a request.
client_port	integer	Port number of the client that sends a request.
unique_query_id	bigint	ID of the normalized SQL statement.
debug_query_id	bigint	ID of the unique SQL statement.
query	text	Normalized SQL (available only on CNs).
start_time	timestamp with time zone	Time when a statement starts.
finish_time	timestamp with time zone	Time when a statement ends.
slow_sql_threshold	bigint	Standard for slow SQL statement execution.
transaction_id	bigint	Transaction ID.
thread_id	bigint	ID of an execution thread.
session_id	bigint	Session ID of a user.
n_soft_parse	bigint	Number of soft parsing times. The value of <b>n_soft_parse</b> plus the value of <b>n_hard_parse</b> may be greater than the value of <b>n_calls</b> because the number of subqueries is not counted in the value of <b>n_calls</b> .
n_hard_parse	bigint	Number of hard parsing times. The value of <b>n_soft_parse</b> plus the value of <b>n_hard_parse</b> may be greater than the value of <b>n_calls</b> because the number of subqueries is not counted in the value of <b>n_calls</b> .
query_plan	text	Statement execution plan.
n_returned_rows	bigint	Number of rows in the result set returned by the SELECT statement.

Name	Type	Description
n_tuples_fetched	bigint	Number of rows randomly scanned.
n_tuples_returned	bigint	Number of rows sequentially scanned.
n_tuples_inserted	bigint	Number of rows inserted.
n_tuples_updated	bigint	Number of rows updated.
n_tuples_deleted	bigint	Number of rows deleted.
n_blocks_fetched	bigint	Number of buffer block access times.
n_blocks_hit	bigint	Number of buffer block hits.
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: $\mu$ s).
cpu_time	bigint	CPU time (unit: $\mu$ s).
execution_time	bigint	Execution time in the executor (unit: $\mu$ s).
parse_time	bigint	SQL parsing time (unit: $\mu$ s)
plan_time	bigint	SQL plan generation time (unit: $\mu$ s).
rewrite_time	bigint	SQL rewriting time (unit: $\mu$ s).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: $\mu$ s).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: $\mu$ s)
data_io_time	bigint	I/O time (unit: $\mu$ s).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, CNs communicate with each other, CNs communicate with customer service ends, and CNs communicate with DN through physical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <code>{"time":xxx, "n_calls":xxx, "size":xxx}</code>

Name	Type	Description
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, CNs communicate with each other, CNs communicate with customer service ends, and CNs communicate with DN through physical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <b>{"time":xxx, "n_calls":xxx, "size":xxx}</b>
net_stream_sen d_info	text	Network status of messages sent through a logical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, DNs of different shards communicate with each other through logical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <b>{"time":xxx, "n_calls":xxx, "size":xxx}</b>
net_stream_rec v_info	text	Network status of messages received through a logical connection, including the time (unit: $\mu$ s), number of calls, and throughput (unit: byte). In a distributed database, DNs of different shards communicate with each other through logical connections. This column can be used to analyze the network overhead of SQL statements in a distributed system. Example: <b>{"time":xxx, "n_calls":xxx, "size":xxx}</b>
lock_count	bigint	Number of locks.
lock_time	bigint	Time required for locking.
lock_wait_count	bigint	Number of lock waits.
lock_wait_time	bigint	Time required for lock waiting.
lock_max_count	bigint	Maximum number of locks.
lwlock_count	bigint	Number of lightweight locks (reserved).
lwlock_wait_co unt	bigint	Number of lightweight lock waits.

Name	Type	Description
lwlock_time	bigint	Time required for lightweight locking (reserved).
lwlock_wait_time	bigint	Time required for lightweight lock waiting.
details	bytea	<p>List of wait events and statement lock events.</p> <p>When the value of the record level is greater than or equal to L0, the list of wait events starts to be recorded. It displays statistics about wait events on the current node. For key events in the kernel, see <a href="#">Table 13-175</a>. Alternatively, view the list of all events in the system in the wait_event_info view. For details about the impact of each transaction lock on services, see <a href="#">LOCK</a>.</p> <p>When the value of <b>track_stmt_stat_level</b> is <b>L2</b>, the list of statement lock events is recorded. The list records events in chronological order. The number of records is affected by the value of the <b>track_stmt_details_size</b> parameter.</p> <p>This column is in binary format and needs to be read by using the parsing function pg_catalog.statement_detail_decode. For details, see <a href="#">Other Functions</a>.</p> <p>Events include:</p> <ul style="list-style-type: none"> <li>• Start locking.</li> <li>• Complete locking.</li> <li>• Start lock waiting.</li> <li>• Complete lock waiting.</li> <li>• Start unlocking.</li> <li>• Complete unlocking.</li> <li>• Start lightweight lock waiting.</li> <li>• Complete lightweight lock waiting.</li> </ul>
is_slow_sql	Boolean	Specifies whether the SQL statement is a slow SQL statement.
trace_id	text	Driver-specific trace ID, which is associated with an application request.

Name	Type	Description
advise	text	Risk information that may cause the SQL statement to be a slow SQL statement. <b>NOTE</b> If the statement is executed in SQLBypass mode (you can run the explain command to view the plan), there is no slow SQL risk and the information in this field may be missing.
parent_unique_sql_id	bigint	Normalized SQL ID of the outer SQL statement. For statements executed in a stored procedure, the value is the normalized SQL ID of the statement that calls the stored procedure. For statements outside the stored procedure, the value is <b>0</b> .

## 13.2.10 Cache and I/O

### 13.2.10.1 STATIO\_USER\_TABLES

**STATIO\_USER\_TABLES** displays I/O status information about all user relationship tables in the namespace.

**Table 13-110** STATIO\_USER\_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in TOAST tables (if any) in the table

Name	Type	Description
tidx_blks_read	bigint	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in the table

### 13.2.10.2 SUMMARY\_STATIO\_USER\_TABLES

**SUMMARY\_STATIO\_USER\_TABLES** displays I/O status information about all user relationship tables in namespaces in the cluster.

**Table 13-111** SUMMARY\_STATIO\_USER\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	numeric	Number of disk blocks read from the table
heap_blks_hit	numeric	Number of cache hits in the table
idx_blks_read	numeric	Number of disk blocks read from indexes in the table
idx_blks_hit	numeric	Number of cache hits in indexes in the table
toast_blks_read	numeric	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	numeric	Number of buffer hits in TOAST tables (if any) in the table
tidx_blks_read	numeric	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	numeric	Number of buffer hits in the TOAST table index (if any) in the table

### 13.2.10.3 GLOBAL\_STATIO\_USER\_TABLES

**GLOBAL\_STATIO\_USER\_TABLES** displays I/O status information about all user relationship tables in namespaces on each node.

**Table 13-112** GLOBAL\_STATIO\_USER\_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in TOAST tables (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in the table

#### 13.2.10.4 STATIO\_USER\_INDEXES

**STATIO\_USER\_INDEXES** displays I/O status information about all user relationship table indexes in namespaces on the current node.

**Table 13-113** STATIO\_USER\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name

Name	Type	Description
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

### 13.2.10.5 SUMMARY\_STATIO\_USER\_INDEXES

**SUMMARY\_STATIO\_USER\_INDEXES** displays I/O status information about all user relationship table indexes in namespaces in the cluster.

**Table 13-114** SUMMARY\_STATIO\_USER\_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

### 13.2.10.6 GLOBAL\_STATIO\_USER\_INDEXES

**GLOBAL\_STATIO\_USER\_INDEXES** displays I/O status information about all user relationship table indexes in namespaces on each node.

**Table 13-115** GLOBAL\_STATIO\_USER\_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index

Name	Type	Description
idx_blks_hit	numeric	Number of cache hits in the index

### 13.2.10.7 STATIO\_USER\_SEQUENCES

**STATIO\_USER\_SEQUENCE** displays I/O status information about all user relationship table sequences in namespaces on the current node.

**Table 13-116** STATIO\_USER\_SEQUENCE columns

Name	Type	Description
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

### 13.2.10.8 SUMMARY\_STATIO\_USER\_SEQUENCES

**SUMMARY\_STATIO\_USER\_SEQUENCES** displays I/O status information about all user relationship table sequences in namespaces in the cluster.

**Table 13-117** SUMMARY\_STATIO\_USER\_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence
blks_hit	numeric	Number of cache hits in the sequence

### 13.2.10.9 GLOBAL\_STATIO\_USER\_SEQUENCES

**GLOBAL\_STATIO\_USER\_SEQUENCES** displays I/O status information about all user relationship table sequences in namespaces on each node.

**Table 13-118** GLOBAL\_STATIO\_USER\_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

### 13.2.10.10 STATIO\_SYS\_TABLES

**STATIO\_SYS\_TABLES** displays I/O status information about all system catalogs in the current namespace.

**Table 13-119** STATIO\_SYS\_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in TOAST tables (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in the table

### 13.2.10.11 SUMMARY\_STATIO\_SYS\_TABLES

**SUMMARY\_STATIO\_SYS\_TABLES** displays I/O status information about all system catalogs in namespaces in the cluster.

**Table 13-120** SUMMARY\_STATIO\_SYS\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	numeric	Number of disk blocks read from the table
heap_blks_hit	numeric	Number of cache hits in the table
idx_blks_read	numeric	Number of disk blocks read from indexes in the table
idx_blks_hit	numeric	Number of cache hits in indexes in the table
toast_blks_read	numeric	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	numeric	Number of buffer hits in TOAST tables (if any) in the table
tidx_blks_read	numeric	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	numeric	Number of buffer hits in the TOAST table index (if any) in the table

### 13.2.10.12 GLOBAL\_STATIO\_SYS\_TABLES

**GLOBAL\_STATIO\_SYS\_TABLES** displays I/O status information about all system catalogs in namespaces on each node.

**Table 13-121** GLOBAL\_STATIO\_SYS\_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table

Name	Type	Description
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in TOAST tables (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in the table

### 13.2.10.13 STATIO\_SYS\_INDEXES

**STATIO\_SYS\_INDEXES** displays the I/O status information about all system catalog indexes in the current namespace.

**Table 13-122** STATIO\_SYS\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

### 13.2.10.14 SUMMARY\_STATIO\_SYS\_INDEXES

**SUMMARY\_STATIO\_SYS\_INDEXES** displays I/O status information about all system catalog indexes in namespaces in the cluster.

**Table 13-123** SUMMARY\_STATIO\_SYS\_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

### 13.2.10.15 GLOBAL\_STATIO\_SYS\_INDEXES

**GLOBAL\_STATIO\_SYS\_INDEXES** displays I/O status information about all system catalog indexes in namespaces on each node.

**Table 13-124** GLOBAL\_STATIO\_SYS\_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

### 13.2.10.16 STATIO\_SYS\_SEQUENCES

**STATIO\_SYS\_SEQUENCES** shows the I/O status information about all the system sequences in the current namespace.

**Table 13-125** STATIO\_SYS\_SEQUENCES columns

Name	Type	Description
relid	oid	OID of the sequence

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

### 13.2.10.17 SUMMARY\_STATIO\_SYS\_SEQUENCES

**SUMMARY\_STATIO\_SYS\_SEQUENCES** displays I/O status information about all system sequences in namespaces in the cluster.

**Table 13-126** SUMMARY\_STATIO\_SYS\_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence
blks_hit	numeric	Number of cache hits in the sequence

### 13.2.10.18 GLOBAL\_STATIO\_SYS\_SEQUENCES

**GLOBAL\_STATIO\_SYS\_SEQUENCES** displays I/O status information about all system sequences in namespaces on each node.

**Table 13-127** GLOBAL\_STATIO\_SYS\_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

### 13.2.10.19 STATIO\_ALL\_TABLES

**STATIO\_ALL\_TABLES** contains I/O statistics about a row of each table (including TOAST tables) in databases.

**Table 13-128** STATIO\_ALL\_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in TOAST tables (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in the table

### 13.2.10.20 SUMMARY\_STATIO\_ALL\_TABLES

**SUMMARY\_STATIO\_ALL\_TABLES** contains I/O statistics about each table (including TOAST tables) in databases in the cluster.

**Table 13-129** SUMMARY\_STATIO\_ALL\_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	numeric	Number of disk blocks read from the table

Name	Type	Description
heap_blks_hit	numeric	Number of cache hits in the table
idx_blks_read	numeric	Number of disk blocks read from indexes in the table
idx_blks_hit	numeric	Number of cache hits in indexes in the table
toast_blks_read	numeric	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	numeric	Number of buffer hits in TOAST tables (if any) in the table
tidx_blks_read	numeric	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	numeric	Number of buffer hits in the TOAST table index (if any) in the table

### 13.2.10.21 GLOBAL\_STATIO\_ALL\_TABLES

**GLOBAL\_STATIO\_ALL\_TABLES** contains I/O statistics about each table (including TOAST tables) in databases on each node.

**Table 13-130** GLOBAL\_STATIO\_ALL\_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from TOAST tables (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in TOAST tables (if any) in the table

Name	Type	Description
tidx_blks_read	bigint	Number of disk blocks read from the TOAST tables index (if any) in the table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in the table

### 13.2.10.22 STATIO\_ALL\_INDEXES

**STATIO\_ALL\_INDEXES** contains one row for each index in the current database, showing I/O statistics about specific indexes.

**Table 13-131** STATIO\_ALL\_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

### 13.2.10.23 SUMMARY\_STATIO\_ALL\_INDEXES

**SUMMARY\_STATIO\_ALL\_INDEXES** contains I/O statistics about each index row in databases in the cluster.

**Table 13-132** SUMMARY\_STATIO\_ALL\_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index

Name	Type	Description
idx_blks_hit	numeric	Number of cache hits in the index

### 13.2.10.24 GLOBAL\_STATIO\_ALL\_INDEXES

**GLOBAL\_STATIO\_ALL\_INDEXES** contains I/O statistics about one row of each index in databases on each node.

**Table 13-133** GLOBAL\_STATIO\_ALL\_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

### 13.2.10.25 STATIO\_ALL\_SEQUENCES

**STATIO\_ALL\_SEQUENCES** contains one row for each sequence in the current database, showing I/O statistics about specific sequences.

**Table 13-134** STATIO\_ALL\_SEQUENCES columns

Name	Type	Description
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

### 13.2.10.26 SUMMARY\_STATIO\_ALL\_SEQUENCES

**SUMMARY\_STATIO\_ALL\_SEQUENCES** contains I/O statistics about one row of each sequence in databases in the cluster.

**Table 13-135** SUMMARY\_STATIO\_ALL\_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence
blks_hit	numeric	Number of cache hits in the sequence

### 13.2.10.27 GLOBAL\_STATIO\_ALL\_SEQUENCES

**GLOBAL\_STATIO\_ALL\_SEQUENCES** contains I/O statistics about one row of each sequence in databases on each node.

**Table 13-136** GLOBAL\_STATIO\_ALL\_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

## 13.2.11 Communication Library

### 13.2.11.1 COMM\_DELAY

**COMM\_DELAY** displays the TCP proxy communications library status for a single DN.

**Table 13-137** COMM\_DELAY columns

Name	Type	Description
node_name	text	Node name

Name	Type	Description
remote_name	text	Name of the peer node
remote_host	text	IP address of the peer node
stream_num	integer	Number of logical stream connections used by the current physical connection
min_delay	integer	Minimum delay of the current physical connection within 1 minute (unit: $\mu$ s) <b>NOTE</b> A negative result is invalid. Wait until the delay status is updated and query again.
average	integer	Average delay of the current physical connection within 1 minute (unit: $\mu$ s)
max_delay	integer	Maximum delay of the current physical connection within 1 minute (unit: $\mu$ s)

### 13.2.11.2 GLOBAL\_COMM\_DELAY

**GLOBAL\_COMM\_DELAY** displays the TCP proxy communications library status for all the DNs.

**Table 13-138** GLOBAL\_COMM\_DELAY columns

Name	Type	Description
node_name	text	Node name
remote_name	text	Name of the peer node
remote_host	text	IP address of the peer node
stream_num	integer	Number of logical stream connections used by the current physical connection
min_delay	integer	Minimum delay of the current physical connection within 1 minute (unit: $\mu$ s) <b>NOTE</b> A negative result is invalid. Wait until the delay status is updated and query again.
average	integer	Average delay of the current physical connection within 1 minute (unit: $\mu$ s)
max_delay	integer	Maximum delay of the current physical connection within 1 minute (unit: $\mu$ s)

### 13.2.11.3 COMM\_RECV\_STREAM

**COMM\_RECV\_STREAM** displays the receiving stream status of all TCP proxy communications libraries on a single DN.

**Table 13-139** COMM\_RECV\_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Stream status
query_id	bigint	<b>debug_query_id</b> corresponding to the stream
pn_id	integer	<b>plan_node_id</b> of the query executed by the stream
send_smp	integer	<b>smpid</b> of the sender of the query executed by the stream
recv_smp	integer	<b>smpid</b> of the receiver of the query executed by the stream
recv_bytes	bigint	Total data volume received by the stream (unit: byte)
time	bigint	Current lifecycle service duration of the stream (unit: ms)
speed	bigint	Average receiving rate of the stream (unit: byte/s)
quota	bigint	Current communication quota value of the stream (unit: byte)
buff_usize	bigint	Current size of the data cache of the stream (unit: byte)

### 13.2.11.4 GLOBAL\_COMM\_RECV\_STREAM

**GLOBAL\_COMM\_RECV\_STREAM** displays the receiving stream status of all TCP proxy communications libraries on all the DNs.

**Table 13-140** GLOBAL\_COMM\_RECV\_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Stream status
query_id	bigint	<b>debug_query_id</b> corresponding to the stream
pn_id	integer	<b>plan_node_id</b> of the query executed by the stream
send_smp	integer	<b>smpid</b> of the sender of the query executed by the stream
recv_smp	integer	<b>smpid</b> of the receiver of the query executed by the stream
recv_bytes	bigint	Total data volume received by the stream (unit: byte)
time	bigint	Current lifecycle service duration of the stream (unit: ms)
speed	bigint	Average receiving rate of the stream (unit: byte/s)
quota	bigint	Current communication quota value of the stream (unit: byte)
buff_usize	bigint	Current size of the data cache of the stream (unit: byte)

### 13.2.11.5 COMM\_SEND\_STREAM

**COMM\_SEND\_STREAM** displays the sending stream status of all TCP proxy communications libraries on a single DN.

**Table 13-141** COMM\_SEND\_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream

Name	Type	Description
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Stream status
query_id	bigint	<b>debug_query_id</b> corresponding to the stream
pn_id	integer	<b>plan_node_id</b> of the query executed by the stream
send_smp	integer	<b>smpid</b> of the sender of the query executed by the stream
recv_smp	integer	<b>smpid</b> of the receiver of the query executed by the stream
send_bytes	bigint	Total data volume sent by the stream (unit: byte)
time	bigint	Current lifecycle service duration of the stream (unit: ms)
speed	bigint	Average sending rate of the stream (unit: byte/s)
quota	bigint	Current communication quota value of the stream (unit: byte)
wait_quota	bigint	Extra time generated when the stream waits for the quota value (unit: ms)

### 13.2.11.6 GLOBAL\_COMM\_SEND\_STREAM

**GLOBAL\_COMM\_SEND\_STREAM** displays the sending stream status of all TCP proxy communications libraries on all the DN's.

**Table 13-142** GLOBAL\_COMM\_SEND\_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID

Name	Type	Description
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Stream status
query_id	bigint	<b>debug_query_id</b> corresponding to the stream
pn_id	integer	<b>plan_node_id</b> of the query executed by the stream
send_smp	integer	<b>smpid</b> of the sender of the query executed by the stream
recv_smp	integer	<b>smpid</b> of the receiver of the query executed by the stream
send_bytes	bigint	Total data volume sent by the stream (unit: byte)
time	bigint	Current lifecycle service duration of the stream (unit: ms)
speed	bigint	Average sending rate of the stream (unit: byte/s)
quota	bigint	Current communication quota value of the stream (unit: byte)
wait_quota	bigint	Extra time generated when the stream waits for the quota value (unit: ms)

### 13.2.11.7 COMM\_STATUS

**COMM\_STATUS** displays the TCP proxy communications library status on a single DN.

**Table 13-143** COMM\_STATUS columns

Name	Type	Description
node_name	text	Node name
rxpck_rate	integer	Receiving rate of the communications library on the node, in byte/s
txpck_rate	integer	Sending rate of the communications library on the node, in byte/s
rxkbyte_rate	bigint	Receiving rate of the communications library on the node, in Kbyte/s
txkbyte_rate	bigint	Sending rate of the communications library on the node, in Kbyte/s

Name	Type	Description
buffer	bigint	Size of the buffer of the Cmailbox
memkbyte_l ibcomm	bigint	Communication memory size of the <b>libcomm</b> process, in bytes
memkbyte_l ibpq	bigint	Communication memory size of the <b>libpq</b> process, in bytes
used_pm	integer	Real-time usage of the <b>postmaster</b> thread
used_sflow	integer	Real-time usage of the <b>gs_sender_flow_controller</b> thread
used_rflow	integer	Real-time usage of the <b>gs_receiver_flow_controller</b> thread
used_rloop	integer	Highest real-time usage among multiple <b>gs_receivers_loop</b> threads
stream	integer	Total number of used logical connections.

### 13.2.11.8 GLOBAL\_COMM\_STATUS

**GLOBAL\_COMM\_STATUS** displays the TCP proxy communications library status on all the DNs.

**Table 13-144** GLOBAL\_COMM\_STATUS columns

Name	Type	Description
node_name	text	Node name
rxpck_rate	integer	Receiving rate of the communications library on the node, in byte/s
txpck_rate	integer	Sending rate of the communications library on the node, in byte/s
rxkbyte_rate	bigint	Receiving rate of the communications library on the node, in Kbyte/s
txkbyte_rate	bigint	Sending rate of the communications library on the node, in Kbyte/s
buffer	bigint	Size of the buffer of the Cmailbox
memkbyte_l ibcomm	bigint	Communication memory size of the <b>libcomm</b> process, in bytes
memkbyte_l ibpq	bigint	Communication memory size of the <b>libpq</b> process, in bytes

Name	Type	Description
used_pm	integer	Real-time usage of the <b>postmaster</b> thread
used_sflow	integer	Real-time usage of the <b>gs_sender_flow_controller</b> thread
used_rflow	integer	Real-time usage of the <b>gs_receiver_flow_controller</b> thread
used_rloop	integer	Highest real-time usage among multiple <b>gs_receivers_loop</b> threads
stream	integer	Total number of used logical connections.

## 13.2.12 Utility

### 13.2.12.1 REPLICATION\_STAT

REPLICATION\_STAT describes information about log synchronization status, such as the locations where the sender sends logs and where the receiver receives logs.

**Table 13-145** REPLICATION\_STAT columns

Name	Type	Description
pid	bigint	PID of the thread.
usesysid	oid	User system ID.
username	name	Username.
application_name	text	Program name.
client_addr	inet	Client address.
client_hostname	text	Client name.
client_port	integer	Port of the client.
backend_start	timestamp with time zone	Start time of the program.
state	text	Log replication state: <ul style="list-style-type: none"> <li>• Catch-up state</li> <li>• Consistent streaming state</li> </ul>
sender_sent_location	text	Location where the sender sends logs.
receiver_write_location	text	Location where the receiver writes logs.

Name	Type	Description
receiver_flush_location	text	Location where the receiver flushes logs.
receiver_replay_location	text	Location where the receiver replays logs.
sync_priority	integer	Priority of synchronous duplication. (0 indicates asynchronization.)
sync_state	text	Synchronization state: <ul style="list-style-type: none"> <li>Asynchronous replication</li> <li>Synchronous replication</li> <li>Potential synchronization</li> </ul>

### 13.2.12.2 GLOBAL\_REPLICATION\_STAT

GLOBAL\_REPLICATION\_STAT displays information about log synchronization status on each node, such as the locations where the sender sends logs and where the receiver receives logs.

**Table 13-146** GLOBAL\_REPLICATION\_STAT columns

Name	Type	Description
node_name	name	Node name.
pid	bigint	PID of the thread.
usesysid	oid	User system ID.
username	name	Username.
application_name	text	Program name.
client_addr	inet	Client address.
client_hostname	text	Client name.
client_port	integer	Port of the client.
backend_start	timestamp with time zone	Start time of the program.
state	text	Log replication state: <ul style="list-style-type: none"> <li>Catch-up state</li> <li>Consistent streaming state</li> </ul>
sender_sent_location	text	Location where the sender sends logs.

Name	Type	Description
receiver_write_location	text	Location where the receiver writes logs.
receiver_flush_location	text	Location where the receiver flushes logs.
receiver_replay_location	text	Location where the receiver replays logs.
sync_priority	integer	Priority of synchronous duplication. ( <b>0</b> indicates asynchronization.)
sync_state	text	Synchronization state: <ul style="list-style-type: none"> <li>Asynchronous replication</li> <li>Synchronous replication</li> <li>Potential synchronization</li> </ul>

### 13.2.12.3 REPLICATION\_SLOTS

**REPLICATION\_SLOTS** displays replication slot information.

**Table 13-147** REPLICATION\_SLOTS columns

Name	Type	Description
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.
slot_type	text	Replication slot type. <ul style="list-style-type: none"> <li><b>physical</b>: physical replication slot.</li> <li><b>logical</b>: logical replication slot.</li> </ul>
datoid	oid	OID of the database where the replication slot resides.
database	name	Name of the database where the replication slot resides.
active	boolean	Determines whether the replication slot is activated. <ul style="list-style-type: none"> <li><b>t</b> (true): yes</li> <li><b>f</b> (false): no</li> </ul>
xmin	xid	XID of the earliest transaction that the database must reserve for the replication slot.

Name	Type	Description
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.
dummy_standby	boolean	Reserved parameter.

### 13.2.12.4 GLOBAL\_REPLICATION\_SLOTS

**GLOBAL\_REPLICATION\_SLOTS** displays information about replicated slots on each node in the cluster.

**Table 13-148** GLOBAL\_REPLICATION\_SLOTS columns

Name	Type	Description
node_name	name	Node name.
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.
slot_type	text	Replication slot type. <ul style="list-style-type: none"> <li>• <b>physical</b>: physical replication slot.</li> <li>• <b>logical</b>: logical replication slot.</li> </ul>
datoid	oid	OID of the database where the replication slot resides.
database	name	Name of the database where the replication slot resides.
active	boolean	Determines whether the replication slot is activated. <ul style="list-style-type: none"> <li>• <b>t</b> (true): yes</li> <li>• <b>f</b> (false): no</li> </ul>
x_min	xid	XID of the earliest transaction that the database must reserve for the replication slot.
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.

Name	Type	Description
dummy_standby	boolean	Reserved parameter.

### 13.2.12.5 PARALLEL\_DECODE\_STATUS

**PARALLEL\_DECODE\_STATUS** displays parallel decoding information about replication slots on the current node. This view needs to be queried on a CN. No result is returned if you query the view on a DN.

**Table 13-149** PARALLEL\_DECODE\_STATUS columns

Name	Type	Description
slot_name	text	Specifies the replication slot name.
parallel_decode_number	integer	Specifies the number of parallel decoder threads of the replication slot.
read_change_queue_length	text	Concatenates the current length of the log queue read by each decoder thread and then outputs the result.
decode_change_queue_length	text	Concatenates the current length of the decoding result queue of each decoder thread and then outputs the result.
reader_lsn	text	Specifies the location of the log read by the current reader thread.
working_txn_cnt	bigint	Specifies the number of transactions that are being concatenated in the current sender thread.
working_txn_memory	bigint	Specifies the total memory occupied by concatenation transactions in the sender thread, in bytes.
decoded_time	timestampz	Specifies the time of the latest WAL decoded by the replication slot.

### 13.2.12.6 GLOBAL\_PARALLEL\_DECODE\_STATUS

**GLOBAL\_PARALLEL\_DECODE\_STATUS** displays parallel decoding information about replication slots on each primary DN in a cluster. The view needs to be queried on a CN. An error is reported when the view is queried on a DN.

**Table 13-150** GLOBAL\_PARALLEL\_DECODE\_STATUS columns

Name	Type	Description
node_name	name	Specifies the node name.
slot_name	text	Specifies the replication slot name.
parallel_decode_num	integer	Specifies the number of parallel decoder threads of the replication slot.
read_change_queue_length	text	Concatenates the current length of the log queue read by each decoder thread and then outputs the result.
decode_change_queue_length	text	Concatenates the current length of the decoding result queue of each decoder thread and then outputs the result.
reader_lsn	text	Specifies the location of the log read by the current reader thread.
working_txn_cnt	bigint	Specifies the number of transactions that are being concatenated in the current sender thread.
working_txn_memory	bigint	Specifies the total memory occupied by concatenation transactions in the sender thread, in bytes.
decoded_time	timestampz	Specifies the time of the latest WAL decoded by the replication slot.

### 13.2.12.7 PARALLEL\_DECODE\_THREAD\_INFO

**PARALLEL\_DECODE\_THREAD\_INFO** displays information about threads that perform parallel decoding on the current node. This view needs to be queried on a CN. No result is returned if you query the view on a DN.

**Table 13-151** PARALLEL\_DECODE\_THREAD\_INFO columns

Name	Type	Description
thread_id	bigint	Thread ID
slot_name	text	Replication slot name
thread_type	text	Thread type (sender, reader, or decoder)
seq_number	integer	Sequence number (starting from 1) of threads of the same type in the current replication slot

### 13.2.12.8 GLOBAL\_PARALLEL\_DECODE\_THREAD\_INFO

**GLOBAL\_PARALLEL\_DECODE\_THREAD\_INFO** displays information about threads that perform parallel decoding on each primary DN in a cluster. The view needs to be queried on a CN. An error is reported when the view is queried on a DN.

**Table 13-152** GLOBAL\_PARALLEL\_DECODE\_THREAD\_INFO columns

Name	Type	Description
node_name	name	Node name
thread_id	bigint	Thread ID
slot_name	text	Replication slot name
thread_type	text	Thread type (sender, reader, or decoder)
seq_number	integer	Sequence number (starting from 1) of threads of the same type in the current replication slot

### 13.2.12.9 BGWRITER\_STAT

**BGWRITER\_STAT** displays statistics about the background writer process's activities.

**Table 13-153** BGWRITER\_STAT columns

Name	Type	Description
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed
checkpoints_req	bigint	Number of requested checkpoints that have been performed
checkpoint_write_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are written to disk (unit: ms)
checkpoint_sync_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are synchronized to disk (unit: ms)
buffers_checkpoint	bigint	Number of buffers written during checkpoints
buffers_clean	bigint	Number of buffers written by the background writer

Name	Type	Description
maxwritten_clean	bigint	Number of times the background writer stopped a cleaning scan because it had written too many buffers
buffers_backend	bigint	Number of buffers written directly by the backend
buffers_backend_fsync	bigint	Number of times the backend had to execute its own fsync call (normally the background writer handles those even when the backend does its own write)
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time at which these statistics were last reset

### 13.2.12.10 GLOBAL\_BGWRITER\_STAT

**GLOBAL\_BGWRITER\_STAT** displays statistics about the background writer process's activities on each node.

**Table 13-154** GLOBAL\_BGWRITER\_STAT columns

Name	Type	Description
node_name	name	Node name
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed
checkpoints_req	bigint	Number of requested checkpoints that have been performed
checkpoint_write_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are written to disk (unit: ms)
checkpoint_sync_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are synchronized to disk (unit: ms)
buffers_checkpoint	bigint	Number of buffers written during checkpoints
buffers_clean	bigint	Number of buffers written by the background writer

Name	Type	Description
maxwritten_clean	bigint	Number of times the background writer stopped a cleaning scan because it had written too many buffers
buffers_backend	bigint	Number of buffers written directly by a backend
buffers_backend_fsync	bigint	Number of times the backend had to execute its own fsync call (normally the background writer handles those even when the backend does its own write)
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time at which these statistics were last reset

### 13.2.12.11 POOLER\_STATUS

**POOLER\_STATUS** is used to query the cache connection status of the pooler module on the local CN.

**Table 13-155** POOLER\_STATUS columns

Name	Type	Description
database	text	Database name
user_name	text	Username
tid	bigint	In non-thread pool logic, this parameter indicates the ID of the thread connected to the CN. In thread pool logic, this parameter indicates the ID of the session connected to the CN.
node_oid	bigint	OID of the node connected
node_name	name	Name of the node connected
in_use	boolean	Whether the connection is currently used. <ul style="list-style-type: none"><li>• <b>t</b> (true): The connection is in use.</li><li>• <b>f</b> (false): The connection is not in use.</li></ul>
node_port	integer	Port number of the node connected

Name	Type	Description
fdsock	bigint	Port file descriptor
remote_pid	bigint	Thread ID of the remote node connected
session_params	text	Session parameter
used_count	bigint	Number of reuse times of a connection
idx	bigint	Logical connection ID of the connected instance node
streamid	bigint	Stream ID corresponding to each logical connection

### 13.2.12.12 GLOBAL\_COMM\_CHECK\_CONNECTION\_STATUS

GLOBAL\_COMM\_CHECK\_CONNECTION\_STATUS displays the connection status of all CNs and all active nodes (CNs and primary DN). The permission control is inherited from the DBE\_PERF schema.

**Table 13-156** GLOBAL\_COMM\_CHECK\_CONNECTION\_STATUS columns

Name	Type	Description
node_name	name	Instance name.
remote_name	name	Name of the peer instance.
remote_host	text	IP address of the peer instance.
remote_port	bigint	Port number of the peer instance.
is_connected	Boolean	<p>Detects the network status between the current instance and the peer instance:</p> <ul style="list-style-type: none"> <li>• <b>t</b> (true): The network between instances is normal.</li> <li>• <b>f</b> (false): The network between instances is abnormal.</li> </ul>
no_error_occur	Boolean	<p>Pooler connection result between the current instance and the peer instance.</p> <ul style="list-style-type: none"> <li>• <b>t</b> (true) indicates that the pooler connection is normal.</li> <li>• <b>f</b> (false) indicates that the pooler connection is abnormal.</li> </ul>

### 13.2.12.13 GLOBAL\_CKPT\_STATUS

GLOBAL\_CKPT\_STATUS displays the information about checkpoints and flushing pages of all instances in the entire cluster.

**Table 13-157** GLOBAL\_CKPT\_STATUS columns

Name	Type	Description
node_name	text	Instance name.
ckpt_redo_point	text	Checkpoint of the current instance.
ckpt_clog_flush_num	bigint	Number of Clog flushing pages from the startup time to the current time.
ckpt_csnlog_flush_num	bigint	Number of CSN log flushing pages from the startup time to the current time.
ckpt_multixact_flush_num	bigint	Number of MultiXact flushing pages from the startup time to the current time.
ckpt_predicate_flush_num	bigint	Number of predicate flushing pages from the startup time to the current time.
ckpt_twophase_flush_num	bigint	Number of two-phase flushing pages from the startup time to the current time.

### 13.2.12.14 GLOBAL\_DOUBLE\_WRITE\_STATUS

GLOBAL\_DOUBLE\_WRITE\_STATUS displays the information about doublewrite files of all instances in the entire cluster.

**Table 13-158** GLOBAL\_DOUBLE\_WRITE\_STATUS columns

Name	Type	Description
node_name	text	Instance name
curr_dwn	bigint	Sequence number of the doublewrite file
curr_start_page	bigint	Start page for restoring the doublewrite file
file_trunc_num	bigint	Number of times that the doublewrite file is reused
file_reset_num	bigint	Number of reset times after the doublewrite file is full
total_writes	bigint	Total number of I/Os of the doublewrite file

Name	Type	Description
low_threshold_writes	bigint	Number of I/Os for writing the doublewrite files with low efficiency (the number of I/O flushing pages at a time is less than 16.)
high_threshold_writes	bigint	Number of I/Os for writing the doublewrite files with high efficiency (the number of I/O flushing pages at a time is more than 421.)
total_pages	bigint	Total number of pages that are flushed to the doublewrite file area
low_threshold_pages	bigint	Number of pages that are flushed with low efficiency
high_threshold_pages	bigint	Number of pages that are flushed with high efficiency
file_id	bigint	ID of the current doublewrite file

### 13.2.12.15 GLOBAL\_PAGEWRITER\_STATUS

**GLOBAL\_PAGEWRITER\_STATUS** displays the information about checkpoints and flushing pages of all instances in the cluster.

**Table 13-159** GLOBAL\_PAGEWRITER\_STATUS columns

Name	Type	Description
node_name	text	Instance name
pgwr_actual_flush_total_num	bigint	Total number of dirty pages flushed from the startup time to the current time
pgwr_last_flush_num	integer	Number of dirty pages flushed in the previous batch
remain_dirty_page_num	bigint	Estimated number of dirty pages that are not flushed
queue_head_page_rec_lsn	text	<b>recovery_lsn</b> of the first dirty page in the dirty page queue of the current instance
queue_rec_lsn	text	<b>recovery_lsn</b> of the dirty page queue of the current instance
current_xlog_important_lsn	text	The write position of Xlogs in the current instance
ckpt_redo_point	text	Checkpoint of the current instance

### 13.2.12.16 GLOBAL\_POOLER\_STATUS

**GLOBAL\_POOLER\_STATUS** is used to query the cache connection status of the pooler modules on all CNs.

**Table 13-160** GLOBAL\_POOLER\_STATUS columns

Name	Type	Description
source_node_name	name	Source node name
database	text	Database name
user_name	text	Username
tid	bigint	In non-thread pool logic, this parameter indicates the ID of the thread connected to the CN. In thread pool logic, this parameter indicates the ID of the session connected to the CN.
node_oid	bigint	OID of the node connected
node_name	name	Name of the node connected
in_use	boolean	Whether the connection is currently used. <ul style="list-style-type: none"><li>• <b>t</b> (true): The connection is in use.</li><li>• <b>f</b> (false): The connection is not in use.</li></ul>
fdsock	bigint	Port file descriptor
remote_pid	bigint	Thread ID of the remote node connected
session_params	text	Session parameter

### 13.2.12.17 GLOBAL\_RECORD\_RESET\_TIME

**GLOBAL\_RECORD\_RESET\_TIME** is used to obtain statistics about the reset (restart, active/standby switchover, and database deletion) time of each node in the cluster.

**Table 13-161** GLOBAL\_RECORD\_RESET\_TIME columns

Name	Type	Description
node_name	text	Node name.
reset_time	timestamp with time zone	Time to be reset.

### 13.2.12.18 GLOBAL\_REDO\_STATUS

GLOBAL\_REDO\_STATUS displays the replaying of logs about instances in the entire cluster.

**Table 13-162** GLOBAL\_REDO\_STATUS columns

Name	Type	Description
node_name	text	Instance name.
redo_start_ptr	bigint	Start point for replaying the instance logs.
redo_start_time	bigint	Start time (UTC) when the instance logs are replayed.
redo_done_time	bigint	End time (UTC) when the instance logs are replayed.
curr_time	bigint	Current time (UTC) of the instance.
min_recovery_point	bigint	Minimum barrier for the current instance log to provide services after the replay is complete.
read_ptr	bigint	Position for reading the instance logs.
last_replayed_read_ptr	bigint	Position for replaying the instance logs.
recovery_done_ptr	bigint	Replay position after the instance is started.
read_xlog_io_counter	bigint	Number of I/Os when the current instance reads and replays logs.
read_xlog_io_total_dur	bigint	Total I/O duration when the current instance reads and replays logs.
read_data_io_counter	bigint	Number of data page I/O reads during replay in the current instance log.
read_data_io_total_dur	bigint	Total I/O duration reads during replay in the current instance log.
write_data_io_counter	bigint	Number of data page I/O writes during replay in the current instance log.
write_data_io_total_dur	bigint	Total I/O duration writes during replay in the current instance log.
process_pending_counter	bigint	Number of synchronization times of log distribution threads during replay in the instance log.
process_pending_total_dur	bigint	Synchronization duration of log distribution threads during replay in the instance log.

Name	Type	Description
apply_counter	bigint	Number of synchronization times of replay threads during replay in the instance log.
apply_total_dur	bigint	Synchronization duration of replay threads during replay in the instance log.
speed	bigint	Log replay rate of the current instance. The value is updated every time when a 256-MB log is replayed. The unit is byte/s.  In a cluster environment, you are advised to run the <b>cm_ctl query -rv</b> command to obtain a more accurate replay speed of the standby node. For details about the <b>cm_ctl</b> command, see "Unified Cluster Management Tool" in <i>Tool Reference</i> .
local_max_ptr	bigint	Maximum number of replay logs received by the localhost after the instance is started.
primary_flush_ptr	bigint	Log point where the host flushes logs to a disk.
worker_info	text	Replay thread information of the instance. If parallel replay is not enabled, the value is <b>NULL</b> .

### 13.2.12.19 GLOBAL\_RECOVERY\_STATUS

GLOBAL\_RECOVERY\_STATUS displays log flow control information about the primary and standby nodes.

**Table 13-163** GLOBAL\_RECOVERY\_STATUS columns

Name	Type	Description
node_name	text	Node name (including the primary and standby nodes).
standby_node_name	text	Name of the standby node.
source_ip	text	IP address of the primary node.
source_port	integer	Port number of the primary node.
dest_ip	text	IP address of the standby node.
dest_port	integer	Port number of the standby node.
current_rto	bigint	Current log flow control time of the standby node (unit: s).

Name	Type	Description
target_rto	bigint	Expected flow control time of the standby node specified by the corresponding GUC parameter (unit: s).
current_sleep_time	bigint	Sleep time (in microseconds) required by the host to achieve the expected RTO.

### 13.2.12.20 CLASS\_VITAL\_INFO

CLASS\_VITAL\_INFO is used to check whether the OIDs of the same table or index are consistent for WDR snapshots.

**Table 13-164** CLASS\_VITAL\_INFO columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Schema name.
relname	name	Table name.
relkind	"char"	Object type. Its value can be: <ul style="list-style-type: none"> <li>• r: ordinary table.</li> <li>• t: TOAST table.</li> <li>• i: index.</li> </ul>

### 13.2.12.21 USER\_LOGIN

USER\_LOGIN records the number of user logins and logouts.

**Table 13-165** USER\_LOGIN columns

Name	Type	Description
node_name	text	Node name
user_name	text	Username
user_id	integer	User OID (Its value is the same as that of <b>oid</b> in <b>pg_authid</b> .)
login_counter	bigint	Number of logins
logout_counter	bigint	Number of logouts

### 13.2.12.22 SUMMARY\_USER\_LOGIN

**SUMMARY\_USER\_LOGIN** records information about user logins and logouts on all CNs.

**Table 13-166** SUMMARY\_USER\_LOGIN columns

Name	Type	Description
node_name	text	Node name
user_name	text	Username
user_id	integer	User OID (Its value is the same as that of <b>oid</b> in <b>pg_authid</b> .)
login_counter	bigint	Number of logins
logout_counter	bigint	Number of logouts

### 13.2.12.23 GLOBAL\_GET\_BGWRITER\_STATUS

**GLOBAL\_GET\_BGWRITER\_STATUS** displays the information about pages flushed by the bgwriter threads of all instances in the entire cluster, number of pages in the candidate buffer chain, and buffer eviction information.

**Table 13-167** GLOBAL\_GET\_BGWRITER\_STATUS columns

Name	Type	Description
node_name	text	Node name.
bgwr_actual_flush_total_num	bigint	Total number of dirty pages flushed by the bgwriter thread from the startup time to the current time.
bgwr_last_flush_num	integer	Number of dirty pages flushed by the bgwriter thread in the previous batch.
candidate_slots	integer	Number of pages in the current candidate buffer chain.
get_buffer_from_list	bigint	Number of times that pages are obtained from the candidate buffer chain during buffer eviction.
get_buffer_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction.

### 13.2.12.24 GLOBAL\_SINGLE\_FLUSH\_DW\_STATUS

**GLOBAL\_SINGLE\_FLUSH\_DW\_STATUS** displays information about doublewrite files eliminated on a single page of all instances in the entire cluster. In the displayed

information, the information before the slash (/) indicates the page flushing status of the first version, and the information after the slash (/) indicates the page flushing status of the second version.

**Table 13-168** GLOBAL\_SINGLE\_FLUSH\_DW\_STATUS columns

Name	Type	Description
node_name	text	Node name.
curr_dwn	text	Sequence number of the doublewrite file.
curr_start_page	text	Start position of the current doublewrite file.
total_writes	text	Total number of data write pages in the current doublewrite file.
file_trunc_num	text	Number of times that the doublewrite file is reused.
file_reset_num	text	Number of reset times after the doublewrite file is full.

### 13.2.12.25 GLOBAL\_CANDIDATE\_STATUS

GLOBAL\_CANDIDATE\_STATUS displays the number of candidate buffers and buffer eviction information of all instances in the database.

**Table 13-169** GLOBAL\_GET\_BGWRITER\_STATUS columns

Name	Type	Description
node_name	text	Node name.
candidate_slots	integer	Number of pages in the candidate buffer chain of the current normal buffer pool.
get_buf_from_list	bigint	Number of times that pages are obtained from the candidate buffer chain during buffer eviction in the current normal buffer pool.
get_buf_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction in the current normal buffer pool.
seg_candidate_slots	integer	Number of pages in the candidate buffer chain of the current segment buffer pool.
seg_get_buf_from_list	bigint	Number of times that pages are obtained from the candidate buffer chain during buffer eviction in the current segment buffer pool.

Name	Type	Description
seg_get_buf_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction in the current segment buffer pool.

## 13.2.13 Lock

### 13.2.13.1 LOCKS

**LOCKS** displays information about locks held by each open transaction.

**Table 13-170** LOCKS columns

Name	Type	Description
locktype	text	Type of the locked object: <b>relation</b> , <b>extend</b> , <b>page</b> , <b>tuple</b> , <b>transactionid</b> , <b>virtualxid</b> , <b>object</b> , <b>userlock</b> , or <b>advisory</b>
database	oid	OID of the database in which the locked object exists. <ul style="list-style-type: none"> <li>The OID is <b>0</b> if the object is a shared object.</li> <li>The OID is <b>NULL</b> if the object is a transaction ID.</li> </ul>
relation	oid	OID of the relationship targeted by the lock. The value is <b>NULL</b> if the object is not a relationship or part of a relationship.
page	integer	Page number targeted by the lock within the relationship. The value is <b>NULL</b> if the object is not a relationship page or row page.
tuple	smallint	Row number targeted by the lock within the page. The value is <b>NULL</b> if the object is not a row.
bucket	integer	Hash bucket number
virtualxid	text	Virtual ID of the transaction targeted by the lock. The value is <b>NULL</b> if the object is not a virtual transaction ID.
transactionid	xid	ID of the transaction targeted by the lock. The value is <b>NULL</b> if the object is not a transaction ID.

Name	Type	Description
classid	oid	OID of the system catalog that contains the object. The value is <b>NULL</b> if the object is not a general database object.
objid	oid	OID of the locked object within its system catalog. The value is <b>NULL</b> if the object is not a general database object.
objsubid	smallint	Column number for a column in the table. The value is <b>0</b> if the object is some other object type. The value is <b>NULL</b> if the object is not a general database object.
virtualtransaction	text	Virtual ID of the transaction holding or awaiting this lock
pid	bigint	Logical ID of the server thread holding or awaiting this lock. The value is <b>NULL</b> if the lock is held by a prepared transaction.
sessionid	bigint	ID of the session holding or awaiting this lock The value is <b>NULL</b> if the lock is held by a prepared transaction.
mode	text	Lock mode held or desired by this thread
granted	boolean	<ul style="list-style-type: none"><li>• The value is <b>TRUE</b> if the lock is a held lock.</li><li>• The value is <b>FALSE</b> if the lock is an awaited lock.</li></ul>
fastpath	boolean	The value is <b>TRUE</b> if the lock is obtained through <b>fast-path</b> , and is <b>FALSE</b> if the lock is obtained through the main lock table.
locktag	text	Information about the lock that the session waits for. It can be parsed using the <b>locktag_decode()</b> function.
global_sessionid	text	Global session ID

### 13.2.13.2 GLOBAL\_LOCKS

**GLOBAL\_LOCKS** displays information about locks held by open transactions on each node.

**Table 13-171** GLOBAL\_LOCKS columns

Name	Type	Description
node_name	name	Node name

Name	Type	Description
locktype	text	Type of the locked object: <b>relation</b> , <b>extend</b> , <b>page</b> , <b>tuple</b> , <b>transactionid</b> , <b>virtualxid</b> , <b>object</b> , <b>userlock</b> , or <b>advisory</b>
database	oid	OID of the database in which the locked object exists. <ul style="list-style-type: none"> <li>The OID is <b>0</b> if the object is a shared object.</li> <li>The OID is <b>NULL</b> if the object is a transaction ID.</li> </ul>
relation	oid	OID of the relationship targeted by the lock. The value is <b>NULL</b> if the object is not a relationship or part of a relationship.
page	integer	Page number targeted by the lock within the relationship. The value is <b>NULL</b> if the object is not a relationship page or row page.
tuple	smallint	Row number targeted by the lock within the page. The value is <b>NULL</b> if the object is not a row.
bucket	integer	Hash bucket ID
virtualxid	text	Virtual ID of the transaction targeted by the lock. The value is <b>NULL</b> if the object is not a virtual transaction ID.
transactionid	xid	ID of the transaction targeted by the lock. The value is <b>NULL</b> if the object is not a transaction ID.
classid	oid	OID of the system catalog that contains the object. The value is <b>NULL</b> if the object is not a general database object.
objid	oid	OID of the locked object within its system catalog. The value is <b>NULL</b> if the object is not a general database object.
objsubid	smallint	Column number for a column in the table. The value is <b>0</b> if the object is some other object type. The value is <b>NULL</b> if the object is not a general database object.
virtualtransaction	text	Virtual ID of the transaction holding or awaiting this lock
pid	bigint	Logical ID of the server thread holding or awaiting this lock. The value is <b>NULL</b> if the lock is held by a prepared transaction.

Name	Type	Description
sessionid	bigint	ID of the session holding or awaiting this lock The value is <b>NULL</b> if the lock is held by a prepared transaction.
global_sessionid	text	Global session ID
mode	text	Lock mode held or desired by this thread
granted	boolean	<ul style="list-style-type: none"><li>The value is <b>TRUE</b> if the lock is a held lock.</li><li>The value is <b>FALSE</b> if the lock is an awaited lock.</li></ul>
fastpath	boolean	The value is <b>TRUE</b> if the lock is obtained through <b>fast-path</b> , and is <b>FALSE</b> if the lock is obtained through the main lock table.
locktag	text	Information about the lock that the session waits for. It can be parsed using the <b>locktag_decode()</b> function.

## 13.2.14 Wait Event

### 13.2.14.1 WAIT\_EVENTS

**WAIT\_EVENTS** displays statistics about wait events on the current node. For key events in the kernel, see [Table 13-175](#). Alternatively, view the list of all events in the system in the **wait\_event\_info** view. For details about the impact of each transaction lock on services, see [LOCK](#).

**Table 13-172** WAIT\_EVENTS columns

Name	Type	Description
nodename	text	Node name
type	text	Event type
event	text	Event name
wait	bigint	Number of waiting times
failed_wait	bigint	Number of waiting failures
total_wait_time	bigint	Total waiting time (unit: $\mu$ s)
avg_wait_time	bigint	Average waiting time (unit: $\mu$ s)
max_wait_time	bigint	Maximum waiting time (unit: $\mu$ s)
min_wait_time	bigint	Minimum waiting time (unit: $\mu$ s)

Name	Type	Description
last_updated	timestamp with time zone	Last time when the event was updated

### 13.2.14.2 GLOBAL\_WAIT\_EVENTS

**GLOBAL\_WAIT\_EVENTS** displays statistics about wait events on each node.

**Table 13-173** GLOBAL\_WAIT\_EVENTS columns

Name	Type	Description
nodename	text	Node name
type	text	Event type
event	text	Event name
wait	bigint	Number of waiting times
failed_wait	bigint	Number of waiting failures
total_wait_time	bigint	Total waiting time (unit: $\mu$ s)
avg_wait_time	bigint	Average waiting time (unit: $\mu$ s)
max_wait_time	bigint	Maximum waiting time (unit: $\mu$ s)
min_wait_time	bigint	Minimum waiting time (unit: $\mu$ s)
last_updated	timestamp with time zone	Last time when the event was updated

### 13.2.14.3 WAIT\_EVENT\_INFO

**WAIT\_EVENT\_INFO** displays the details about wait events.

**Table 13-174** WAIT\_EVENT\_INFO columns

Name	Type	Description
module	text	Name of the module an event belongs to.
type	text	Event type.
event	text	Event name.

**Table 13-175** List of wait events

Module Category	Event Category	Event	Description
Lock	Wait event	acquire lock	Waits for locking until the locking succeeds or times out.
SharedMemory	LWLock event	ShmemIndex Lock	Protects the primary index table, a hash table, in shared memory.
Shared buffer	LWLock event	BufMappingLock	Protects operations on a shared-buffer mapping table.
Lmgr	LWLock event	LockMgrLock	Protects the information about a common lock structure.
LWLock	Wait event	acquire lwlock	Waits for a lightweight lock.
I/O	Wait event	wait io	Waits for I/O completion.
COMM	Wait event	wait cmd	Waits for reading network communication packets to complete.
COMM	Wait event	wait pooler get conn	Waits for pooler to obtain connections.
COMM	Wait event	wait pooler abort conn	Waits for pooler to terminate connections.
COMM	Wait event	wait pooler clean conn	Waits for pooler to clear connections.
COMM	Wait event	get conn	Obtains connections to other nodes.
COMM	Wait event	set cmd	Waits for running the <b>SET</b> , <b>RESET</b> , or <b>TRANSACTION BLOCK LEVEL</b> command on the connection.
COMM	Wait event	cancel query	Cancels the SQL statement that is being executed through a connection.
COMM	Wait event	stop query	Stops the query that is being executed through a connection.
COMM	Wait event	wait node	Waits for receiving data through the connection to a node.
COMM	Wait event	flush data	Waits for sending data to other nodes in the network.
COMM	Wait event	stream get conn	Waits for establishing connections to consumer nodes when the stream flow is initialized.

Module Category	Event Category	Event	Description
COMM	Wait event	wait producer ready	Waits for every producer to get ready when the stream flow is initialized.
Stream	Wait event	synchronize quit	Waits for the threads in the stream thread group to exit when the stream plan ends.
Stream	Wait event	wait stream group destroy	Waits for destroying the stream node group when the stream plan ends.
Transaction	Wait event	wait transaction sync	Waits for transaction synchronization.
Transaction	Wait event	wait data sync	Waits for the completion of data page synchronization to the standby node.
Transaction	Wait event	wait data sync queue	Waits for putting the data pages that are in the row-store into the synchronization queue.
Transaction	LWLock event	OidGenLock	Prevents different threads from generating the same OID.
Transaction	LWLock event	XidGenLock	Prevents two transactions from obtaining the same XID.
Transaction	LWLock event	ProcArrayLock	Prevents concurrent access to or concurrent modification on the ProcArray shared arrays.
Transaction	LWLock event	SubtransControlLock	Prevents concurrent access to or concurrent modification on the sub-transaction control data structure.
Transaction	LWLock event	MultiXactGenLock	Allocates a unique MultiXact ID in serial mode.
Transaction	LWLock event	TwoPhaseStateLock	Prevents concurrent access to or concurrent modification on two-phase information sharing arrays.
Transaction	LWLock event	SerializableXactHashLock	Prevents concurrent read/write or concurrent write/write on a sharing structure for serializable transactions.
Transaction	LWLock event	SerializableFinishedListLock	Prevents concurrent read/write or concurrent write/write on a shared linked list for completed serial transactions.
Transaction	LWLock event	SerializablePredicateListLock	Protects a linked list of serializable transactions that have locks.

Module Category	Event Category	Event	Description
Transaction	LWLock event	PredicateLock MgrLock	Protects the information about a lock structure that has serializable transactions.
Transaction	LWLock event	OldSerXid SLRU lwlock	Protects SLRU buffers of old transaction IDs.
Transaction	LWLock event	OldSerXidLock	Protects a structure that records serializable transactions that have conflicts.
Transaction	Lock event	transactionid	Adds a lock to a transaction ID.
Transaction	Lock event	virtualxid	Adds a lock to a virtual transaction ID.
Checkpoint	LWLock event	CheckpointLock	Prevents multi-checkpoint concurrent execution.
Checkpoint	LWLock event	CheckpointComLock	Sends file flush requests to a checkpointer. The request structure needs to be inserted to a request queue in serial mode.
Analyze	LWLock event	AutoanalyzeLock	Obtains and releases resources related to a task that allows for autoanalyze execution.
Vacuum	LWLock event	BtreeVacuumLock	Prevents VACUUM from clearing pages that are being used by B-tree indexes.
Vacuum	LWLock event	AutovacuumLock	Accesses the autovacuum worker array in serial mode.
Vacuum	LWLock event	AutovacuumScheduleLock	Distributes tables requiring VACUUM in serial mode.
Autovacuum	LWLock event	AutovacuumLock	Protects the autovacuum shared memory structure.
Autovacuum	LWLock event	AutovacuumScheduleLock	Protects the information about autovacuum workers.
Autoanalyze	LWLock event	AutoanalyzeLock	Protects the <i>autoAnalyzeFreeProcess</i> variable and ensures that no more than 10 autoanalyze threads are running at the same time.
WAL	Wait event	wait wal sync	Waits for the completion of WAL synchronization from the specified LSN to the standby node.

Module Category	Event Category	Event	Description
WAL	I/O event	WALBootstrapSync	Flushes an initialized WAL file to a disk during database initialization.
WAL	I/O event	WALBootstrapWrite	Writes an initialized WAL file during database initialization.
WAL	I/O event	WALCopyRead	Reads operation generated when an existing WAL file is read for replication after archiving and restoration.
WAL	I/O event	WALCopySync	Flushes a replicated WAL file to a disk after archiving and restoration.
WAL	I/O event	WALCopyWrite	Writes operation generated when an existing WAL file is read for replication after archiving and restoration.
WAL	I/O event	WALInitSync	Flushes a newly initialized WAL file to a disk during log reclaiming or writing.
WAL	I/O event	WALInitWrite	Initializes a newly created WAL file to 0 during log reclaiming or writing.
WAL	I/O event	WALRead	Reads data from Xlogs during redo operations on two-phase files.
WAL	I/O event	WALSyncMethodAssign	Flushes all open WAL files to a disk.
WAL	I/O event	WALWrite	Writes a WAL file.
WAL	I/O event	LOGCTRL_SLEEP	Collects statistics on the number of stream control times and the sleep time of log stream control.
WAL	LWLock event	RcvWriteLock	Prevents concurrent call of WalDataRcvWrite.
WAL	LWLock event	WALBufMappingLock	An exclusive (X) lock needs to be added when the next page of an Xlog buffer is initialized.
WAL	LWLock event	WALInsertLock	Prevents multiple programs from writing data to the same Xlog buffer at the same time.
WAL	LWLock event	WALWriteLock	Prevents concurrent WAL write.
Relation	LWLock event	SInvalReadLock	Prevents concurrent execution with invalid message deletion.
Relation	LWLock event	SInvalWriteLock	Prevents concurrent execution with invalid message write and deletion.

Module Category	Event Category	Event	Description
Relation	LWLock event	RelCacheInitLock	Adds a lock before any operations are performed on the <b>init</b> file when messages are invalid.
Relation	LWLock event	TablespaceCreateLock	Checks whether a tablespace already exists.
Relation	LWLock event	RelfilenodeReuseLock	Prevents the link to a reused column attribute file from being canceled by mistake.
Relation	Lock event	relation	Adds a lock to a table.
Relation	Lock event	extend	Adds a lock to a table being scaled out.
Relation	Lock event	partition	Adds a lock to a partitioned table.
Relation	Lock event	partition_seq	Adds a lock to a partition of a partitioned table.
WLM	Wait event	wait active statement	Waits for active statements.
WLM	Wait event	wait memory	Waits for free memory.
DDL/DCL	Wait event	create index	Waits for the completion of index creation.
DDL/DCL	Wait event	analyze	Waits for analysis completion.
DDL/DCL	Wait event	vacuum	Waits for the completion of the VACUUM operation.
DDL/DCL	LWLock event	DelayDDLlock	Prevents concurrent DDL operations.
DDL/DCL	Wait event	vacuum full	Waits for the completion of the VACUUM FULL operation.
Executor	Wait event	Sort	Waits for the completion of tuple sorting.
Executor	Wait event	Sort - write file	Writes sorted data to a file temporarily since the memory is limited during merge sort.
Executor	Wait event	Material	Waits for tuple materialization.

Module Category	Event Category	Event	Description
Executor	Wait event	Material - write file	Waits for writing a materialized tuple to a file.
Executor	Wait event	HashJoin - build hash	Waits until a hash table is created when a hash join is executed.
Executor	Wait event	HashJoin - write file	Waits for writing the hash result of a tuple to a disk when a hash join is executed.
Executor	Wait event	HashAgg - build hash	Waits until a hash table is created when a hash aggregate is executed.
Executor	Wait event	HashAgg - write file	Waits for writing the hash result of a tuple to a disk when a hash aggregate is executed.
Executor	Wait event	HashSetop - build hash	Waits until a hash table is created when an OP operation is performed using the hash algorithm.
Executor	Wait event	HashSetop - write file	Waits for writing the hash result of a tuple to a disk when an OP operation is performed using the hash algorithm.
Executor	Wait event	wait sync consumer next step	Waits for the stream consumer to perform the next step.
Executor	Wait event	wait sync producer next step	Waits for the stream producer to perform the next step.
GTM	Wait event	gtm connect	Waits for connecting to GTM.
GTM	Wait event	gtm reset xmin	Waits for GTM to reset the minimum transaction ID.
GTM	Wait event	gtm get xmin	Waits for obtaining the minimum transaction ID from GTM.
GTM	Wait event	gtm get gxid	Waits for obtaining the global transaction ID from GTM during transaction startup.
GTM	Wait event	gtm get csn	Waits for obtaining the CSN from GTM during transaction startup.
GTM	Wait event	gtm get snapshot	Waits for obtaining snapshots from GTM during transaction startup.
GTM	Wait event	gtm begin trans	Waits for GTM to start a transaction.

Module Category	Event Category	Event	Description
GTM	Wait event	gtm commit trans	Waits for GTM to commit a transaction.
GTM	Wait event	gtm rollback trans	Waits for GTM to roll back transactions.
GTM	Wait event	gtm start prepare trans	Waits for GTM to complete the first phase during two-phase commit.
GTM	Wait event	gtm prepare trans	Waits for GTM to complete the second phase during two-phase commit.
GTM	Wait event	gtm open sequence	Waits for GTM to create a sequence.
GTM	Wait event	gtm close sequence	Waits for GTM to complete the ALTER SEQUENCE operation.
GTM	Wait event	gtm set sequence val	Waits for GTM to set a sequence value.
GTM	Wait event	gtm drop sequence	Waits for GTM to delete a sequence.
GTM	Wait event	gtm rename sequence	Waits for GTM to rename a sequence.
GTM	LWLock event	GTMHostInfo Lock	Protects GTM information.
Temp File	I/O event	BufFileRead	Reads data from a temporary file to a specified buffer.
Temp File	I/O event	BufFileWrite	Writes the content of a specified buffer to a temporary file.
Pg_control	I/O event	ControlFileRead	Reads the <b>pg_control</b> file, mainly during database startup, checkpoint execution, and primary/standby verification.
Pg_control	I/O event	ControlFileSync	Flushes the <b>pg_control</b> file to a disk, mainly during database initialization.
Pg_control	I/O event	ControlFileSyncUpdate	Flushes the <b>pg_control</b> file to a disk, mainly during database startup, checkpoint execution, and primary/standby verification.
Pg_control	I/O event	ControlFileWrite	Writes the <b>pg_control</b> file, mainly during database initialization.

Module Category	Event Category	Event	Description
Pg_control	I/O event	ControlFileWriteUpdate	Updates the <b>pg_control</b> file, mainly during database startup, checkpoint execution, and primary/standby verification.
Pg_control	LWLock event	ControlFileLock	Prevents concurrent read/write or concurrent write/write on the <b>pg_control</b> file.
File operation	I/O event	CopyFileRead	Reads a file during file copying.
File operation	I/O event	CopyFileWrite	Writes a file during file copying.
File operation	I/O event	DataFileExtend	Writes a file during file name extension.
Table data file	I/O event	DataFileImmediateSync	Flushes a table data file to a disk immediately.
Table data file	I/O event	DataFilePrefetch	Reads a table data file asynchronously.
Table data file	I/O event	DataFileRead	Reads a table data file synchronously.
Table data file	I/O event	DataFileSync	Synchronizes a table data file to a disk.
Table data file	I/O event	DataFileTruncate	Truncates a table data file.
Table data file	I/O event	DataFileWrite	Writes a table data file.
Table data file	LWLock event	SyncScanLock	Determines the start position of a relfilenode during heap scanning.
Table data file	LWLock event	RelationMappingLock	Waits for the mapping file between system catalogs and storage locations to be updated.
metadata	LWLock event	MetaCacheSweepLock	Adds a lock when metadata is cyclically washed out.
postmaster.pid	I/O event	LockFileAddToDataDirRead	Reads the <b>postmaster.pid</b> file.
postmaster.pid	I/O event	LockFileAddToDataDirSync	Flushes the <b>postmaster.pid</b> file to a disk.

Module Category	Event Category	Event	Description
postmaster.pid	I/O event	LockFileAddT oDataDirWrit e	Writes PID information into the <b>postmaster.pid</b> file.
Pid File	I/O event	LockFileCreat eRead	Reads the LockFile file <b>%s.lock</b> .
Pid File	I/O event	LockFileCreat eSync	Flushes the LockFile file <b>%s.lock</b> to a disk.
Pid File	I/O event	LockFileCreat eWRITE	Writes PID information into the LockFile file <b>%s.lock</b> .
System catalog mapping file	I/O event	RelationMap Read	Reads the mapping file between system catalogs and storage locations.
System catalog mapping file	I/O event	RelationMap Sync	Flushes the mapping file between system catalogs and storage locations to a disk.
System catalog mapping file	I/O event	RelationMap Write	Writes the mapping file between system catalogs and storage locations.
Streaming replication	I/O event	ReplicationSl otRead	Reads a streaming replication slot file during a restart.
Streaming replication	I/O event	ReplicationSl otRestoreSyn c	Flushes a streaming replication slot file to a disk.
Streaming replication	I/O event	ReplicationSl otSync	Flushes a temporary streaming replication slot file to a disk during checkpoint execution.
Streaming replication	I/O event	ReplicationSl otWrite	Writes a temporary streaming replication slot file during checkpoint execution.
Streaming replication	LWLock event	ReplicationSl otAllocationL ock	Allocates a replication slot.
Streaming replication	LWLock event	ReplicationSl otControlLoc k	Detects replication slot name conflicts and identifies replication slots that can be allocated.

Module Category	Event Category	Event	Description
Clog	I/O event	SLRUFlushSync	Flushes the <b>pg_clog</b> file to a disk, mainly during checkpoint execution and database shutdown.
Clog	I/O event	SLRURead	Reads the <b>pg_clog</b> file.
Clog	I/O event	SLRUSync	Writes dirty pages into the <b>pg_clog</b> file, and flushes the file to a disk, mainly during checkpoint execution and database shutdown.
Clog	I/O event	SLRUWrite	Writes the <b>pg_clog</b> file.
Clog	LWLock event	CLogControlLock	Prevents concurrent access to or concurrent modification on the Clog control data structure.
Clog	LWLock event	MultiXactOffsetControlLock	Prevents concurrent read/write or concurrent write/write on <b>pg_multixact/offset</b> .
Clog	LWLock event	MultiXactMemberControlLock	Prevents concurrent read/write or concurrent write/write on <b>pg_multixact/members</b> .
timelinehistory	I/O event	TimelineHistoryRead	Reads the <b>timelinehistory</b> file during database startup.
timelinehistory	I/O event	TimelineHistorySync	Flushes the <b>timelinehistory</b> file to a disk during database startup.
timelinehistory	I/O event	TimelineHistoryWrite	Writes the <b>timelinehistory</b> file.
pg_twophase	I/O event	TwophaseFileRead	Reads the <b>pg_twophase</b> file, mainly during two-phase transaction commit and restoration.
pg_twophase	I/O event	TwophaseFileSync	Flushes the <b>pg_twophase</b> file to a disk, mainly during two-phase transaction commit and restoration.
pg_twophase	I/O event	TwophaseFileWrite	Writes the <b>pg_twophase</b> file, mainly during two-phase transaction commit and restoration.
Cluster	LWLock event	NodeTableLock	Protects a shared structure that stores CNs and DNs information.
Concurrency	LWLock event	PoolerLock	Prevents two threads from simultaneously obtaining the same connection from a connection pool.

Module Category	Event Category	Event	Description
Concurren- cy	LWLock event	AsyncCtlLock	Prevents concurrent access to or concurrent modification on the sharing notification status.
Concurren- cy	LWLock event	AsyncQueueL ock	Prevents concurrent access to or concurrent modification on the sharing notification queue.
Double write	I/O event	DoubleWriteF ileWrite	Writes pages to a doublewrite file during the doublewrite process.
Double write	I/O event	DoubleWriteF ileRead	Reads a doublewrite file during restoration for a halfwrite.
Statistics file	LWLock event	FileStatLock	Protects a data structure that stores statistics file information.
Master- slave replicatio n	LWLock event	SyncRepLock	Protects Xlog synchronization information during primary/standby replication.
Master- slave replicatio n	LWLock event	ReplicationSl otAllocationL ock	Adds a lock when a primary node allocates streaming replication slots during primary/standby replication.
Master- slave replicatio n	LWLock event	ReplicationSl otControlLoc k	Prevents concurrent update of streaming replication slot status during primary/standby replication.
Master- slave replicatio n	LWLock event	LsnXlogChkFil eLock	Serially updates the Xlog flush points for primary and standby nodes recorded in a specific structure.
Master- slave replicatio n	LWLock event	DataSyncRep Lock	Protects data page synchronization information during primary/standby replication.
Speed up the cluster	LWLock event	SearchServer LibLock	Adds a lock on the file read operation when a specific dynamic library is initially loaded in GPU-accelerated scenarios.
Resource manage	LWLock event	ResourcePool HashLock	Prevents concurrent access to or concurrent modification on a resource pool table, a hash table.

Module Category	Event Category	Event	Description
Resource manage	LWLock event	WorkloadStatHashLock	Prevents concurrent access to or concurrent modification on a hash table that contains SQL requests from the CN side.
Resource manage	LWLock event	WorkloadIoStatHashLock	Prevents concurrent access to or concurrent modification on a hash table that contains I/O information of the current DN.
Resource manage	LWLock event	WorkloadCGroupHashLock	Prevents concurrent access to or concurrent modification on a hash table that contains Cgroup information.
Resource manage	LWLock event	WorkloadRecordLock	Prevents concurrent access to or concurrent modification on a hash table that contains requests received by CNs during adaptive memory management.
Resource manage	LWLock event	WorkloadIOUtilLock	Protects a structure that records <b>iostat</b> and CPU load information.
Resource manage	LWLock event	WorkloadNodeGroupLock	Prevents concurrent access to or concurrent modification on a hash table that contains node group information in memory.
OBS	LWLock event	OBSGetPathLock	Prevents concurrent read/write or concurrent write/write on an OBS path.
OBS	LWLock event	OBSRuntimeLock	Obtains environment variables, for example, <i>GAUSSHOME</i> .
MPP is compatible with ORACLE scheduled task function	LWLock event	JobShmemLock	Used to protect global variables in the shared memory that is periodically read during a scheduled job where MPP is compatible with Oracle Database.
Operator history information statistics	LWLock event	OperatorRealTLock	Prevents concurrent access to or concurrent modification on a global structure that contains real-time data at the operator level.

Module Category	Event Category	Event	Description
Operator history information statistics	LWLock event	OperatorHistLock	Prevents concurrent access to or concurrent modification on a global structure that contains historical data at the operator level.
query history information statistics	LWLock event	SessionRealTLock	Prevents concurrent access to or concurrent modification on a global structure that contains real-time data at the query level.
query history information statistics	LWLock event	SessionHistLock	Prevents concurrent access to or concurrent modification on a global structure that contains historical data at the query level.
query history information statistics	LWLock event	WaitCountHashLock	Protects a shared structure in user statement counting scenarios.
barrier	LWLock event	BarrierLock	Ensures that only one thread is creating a barrier at a time.
CSN	LWLock event	CSNBufMappingLock	Protects CSN pages.
instrumentation	LWLock event	UniqueSQLMappingLock	Protects a unique SQL hash table.
instrumentation	LWLock event	InstrUserLock	Protects InstrUserHTAB.
instrumentation	LWLock event	PercentileLock	Protects global percentile buffers.
instrumentation	LWLock event	InstrWorkloadLock	Protects a workload transaction hash table.
Pgproc	LWLock event	Pgproclwlock	Protects the PGPROC structure.
Async buffer	LWLock event	AsyncCtlLock	Protects asynchronization buffers.
MultiXact	LWLock event	MultiXactOffsetlwlock	Protects SLRU buffers of a MultiXact offset.
MultiXact	LWLock event	MultiXactMemberlwlock	Protects SLRU buffer of a MultiXact member.

Module Category	Event Category	Event	Description
CBM	LWLock event	CBMParseXlogLock	Protects the lock used when CBM parses Xlogs.
BadBlock	LWLock event	BadBlockStat HashLock	Protects the hash table <b>global_bad_block_stat</b> .
Page	Lock event	page	Adds a lock to a table page.
Tuple	Lock event	tuple	Adds a lock to a tuple on a page.
object	Lock event	object	Adds a lock to an object.
user	Lock event	userlock	Adds a lock to a user.
advisor	Lock event	advisory	Adds an advisory lock.
ODBC	LWLock event	ExtensionConnectorLibLock	Adds a lock when a specific dynamic library is loaded or uninstalled in ODBC connection initialization scenarios.
Undo	Wait event	wait fetch undo record	Waits for reading the target undo record.
Heap	Wait event	wait heap hot search buffer	Waits for reading the Astore tuple that meets the snapshot requirements through the hot link.
LWLock	Wait event	wait exclusive lwlock	The starvation prevention mechanism is triggered. A new lightweight lock request waits for the previously blocked lightweight lock to be obtained.

## 13.2.15 Configuration

### 13.2.15.1 CONFIG\_SETTINGS

**CONFIG\_SETTINGS** displays information about parameters of the running database.

**Table 13-176** CONFIG\_SETTINGS columns

Name	Type	Description
name	text	Parameter name

Name	Type	Description
setting	text	Current parameter value
unit	text	Implicit unit of the parameter
category	text	Logical group of the parameter
short_desc	text	Brief description of the parameter
extra_desc	text	Detailed description of the parameter
context	text	Context required to set the parameter value, including <b>internal</b> , <b>postmaster</b> , <b>sighup</b> , <b>backend</b> , <b>superuser</b> , and <b>user</b>
vartype	text	Parameter type, including <b>bool</b> , <b>enum</b> , <b>integer</b> , <b>real</b> , or <b>string</b>
source	text	Method of assigning the parameter value
min_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is <b>null</b> .
max_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is <b>null</b> .
enumvals	text[]	Valid values of an enum parameter. If the parameter type is not enum, the value of this column is <b>null</b> .
boot_val	text	Default parameter value used upon the database startup
reset_val	text	Default parameter value used upon the database reset
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is <b>null</b> .
sourceline	integer	Row number in the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is <b>null</b> .

### 13.2.15.2 GLOBAL\_CONFIG\_SETTINGS

**GLOBAL\_CONFIG\_SETTINGS** displays information about parameters of running databases on each node.

**Table 13-177** GLOBAL\_CONFIG\_SETTINGS columns

Name	Type	Description
node_name	text	Node name
name	text	Parameter name
setting	text	Current parameter value
unit	text	Implicit unit of the parameter
category	text	Logical group of the parameter
short_desc	text	Brief description of the parameter
extra_desc	text	Detailed description of the parameter
context	text	Context required to set the parameter value, including <b>internal</b> , <b>postmaster</b> , <b>sighup</b> , <b>backend</b> , <b>superuser</b> , and <b>user</b>
vartype	text	Parameter type, including <b>bool</b> , <b>enum</b> , <b>integer</b> , <b>real</b> , or <b>string</b>
source	text	Method of assigning the parameter value
min_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is <b>null</b> .
max_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is <b>null</b> .
enumvals	text[]	Valid values of an enum parameter. If the parameter type is not enum, the value of this column is <b>null</b> .
boot_val	text	Default parameter value used upon the database startup
reset_val	text	Default parameter value used upon the database reset
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is <b>null</b> .
sourceline	integer	Row number in the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is <b>null</b> .

## 13.2.16 Operator

### 13.2.16.1 OPERATOR\_HISTORY\_TABLE

OPERATOR\_HISTORY\_TABLE displays records about operators of completed jobs. Data is dumped from the kernel to this system view.

**Table 13-178** OPERATOR\_HISTORY\_TABLE columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution.
pid	bigint	Backend thread ID.
plan_node_id	integer	Plan node ID of the execution plan of a query.
plan_node_name	text	Name of the operator corresponding to the plan node ID.
start_time	timestamp with time zone	Time when the operator starts to process the first data record.
duration	bigint	Total execution time of the operator (unit: ms).
query_dop	integer	DOP of the operator.
estimated_rows	bigint	Number of rows estimated by the optimizer.
tuple_processed	bigint	Number of elements returned by the operator.
min_peak_memory	integer	Minimum peak memory used by the operator on all DN's (unit: MB).
max_peak_memory	integer	Maximum peak memory used by the operator on all DN's (unit: MB).
average_peak_memory	integer	Average peak memory used by the operator on all DN's (unit: MB).
memory_skew_percent	integer	Memory usage skew of the operator among each DN.
min_spill_size	integer	Minimum spilled data among all DN's when a spill occurs (unit: MB; default value: 0).
max_spill_size	integer	Maximum spilled data among all DN's when a spill occurs (unit: MB; default value: 0).
average_spill_size	integer	Average spilled data among all DN's when a spill occurs (unit: MB; default value: 0).
spill_skew_percent	integer	DN spill skew when a spill occurs.
min_cpu_time	bigint	Minimum execution time of the operator on all DN's (unit: ms).

Name	Type	Description
max_cpu_time	bigint	Maximum execution time of the operator on all DNs (unit: ms).
total_cpu_time	bigint	Total execution time of the operator on all DNs (unit: ms).
cpu_skew_percent	integer	Skew of the execution time among DNs.
warning	text	Warning. The following warnings are displayed: <ul style="list-style-type: none"> <li>• Sort/SetOp/HashAgg/HashJoin spill</li> <li>• Spill file size large than 256MB</li> <li>• Broadcast size large than 100MB</li> <li>• Early spill</li> <li>• Spill times is greater than 3</li> <li>• Spill on memory adaptive</li> <li>• Hash table conflict</li> </ul>

### 13.2.16.2 OPERATOR\_HISTORY

OPERATOR\_HISTORY displays information about operators of the jobs that are executed by the current user on the current CN. Data in the kernel is cleared every 3 minutes.

### 13.2.16.3 OPERATOR\_RUNTIME

OPERATOR\_RUNTIME displays information about operators of the jobs that are being executed by the current user.

**Table 13-179** OPERATOR\_RUNTIME columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Backend thread ID
plan_node_id	integer	Plan node ID of the execution plan of a query
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator (unit: ms)

Name	Type	Description
status	text	Execution status of the current operator. Its value can be <b>finished</b> or <b>running</b> .
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on all DNs (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on all DNs (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on all DNs (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among DNs
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs (unit: MB; default value: <b>0</b> )
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs (unit: MB; default value: <b>0</b> )
average_spill_size	integer	Average spilled data among all DNs when a spill occurs (unit: MB; default value: <b>0</b> )
spill_skew_percent	integer	DN spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on all DNs (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on all DNs (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on all DNs (unit: ms)
cpu_skew_percent	integer	Skew of the execution time among DNs

Name	Type	Description
warning	text	Warning. The following warnings are displayed: <ul style="list-style-type: none"><li>• Sort/SetOp/HashAgg/HashJoin spill</li><li>• Spill file size large than 256MB</li><li>• Broadcast size large than 100MB</li><li>• Early spill</li><li>• Spill times is greater than 3</li><li>• Spill on memory adaptive</li><li>• Hash table conflict</li></ul>

#### 13.2.16.4 GLOBAL\_OPERATOR\_HISTORY

**GLOBAL\_OPERATOR\_HISTORY** displays the records about operators after jobs are executed by the current user on all CNs.

**Table 13-180** GLOBAL\_OPERATOR\_HISTORY columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Backend thread ID
plan_node_id	integer	Plan node ID of the execution plan of a query
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator (unit: ms)
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on all DN's (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on all DN's (unit: MB)

Name	Type	Description
average_peak_memory	integer	Average peak memory used by the operator on all DNs (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among DNs
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs (unit: MB; default value: 0)
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs (unit: MB; default value: 0)
average_spill_size	integer	Average spilled data among all DNs when a spill occurs (unit: MB; default value: 0)
spill_skew_percent	integer	DN spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on all DNs (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on all DNs (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on all DNs (unit: ms)
cpu_skew_percent	integer	Skew of the execution time among DNs
warning	text	Warning. The following warnings are displayed: 1. Sort/SetOp/HashAgg/HashJoin spill 2. Spill file size large than 256MB 3. Broadcast size large than 100MB 4. Early spill 5. Spill times is greater than 3 6. Spill on memory adaptive 7. Hash table conflict

### 13.2.16.5 GLOBAL\_OPERATOR\_HISTORY\_TABLE

GLOBAL\_OPERATOR\_HISTORY\_TABLE displays the records about operators of completed jobs on all CNs. Data is dumped from the kernel to the system catalog GS\_WLM\_OPERATOR\_INFO.

### 13.2.16.6 GLOBAL\_OPERATOR\_RUNTIME

GLOBAL\_OPERATOR\_RUNTIME displays information about operators of the jobs that are being executed by the current user on all CNs.

**Table 13-181** GLOBAL\_OPERATOR\_RUNTIME columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Backend thread ID
plan_node_id	integer	Plan node ID of the execution plan of a query
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator (unit: ms)
status	text	Execution status of the current operator. Its value can be <b>finished</b> or <b>running</b> .
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on all DNs (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on all DNs (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on all DNs (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among DNs
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs (unit: MB; default value: <b>0</b> )
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs (unit: MB; default value: <b>0</b> )
average_spill_size	integer	Average spilled data among all DNs when a spill occurs (unit: MB; default value: <b>0</b> )
spill_skew_percent	integer	DN spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on all DNs (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on all DNs (unit: ms)

Name	Type	Description
total_cpu_time	bigint	Total execution time of the operator on all DNs (unit: ms)
cpu_skew_percent	integer	Skew of the execution time among DNs
warning	text	Warning. The following warnings are displayed: <ul style="list-style-type: none"><li>• Sort/SetOp/HashAgg/HashJoin spill</li><li>• Spill file size large than 256MB</li><li>• Broadcast size large than 100MB</li><li>• Early spill</li><li>• Spill times is greater than 3</li><li>• Spill on memory adaptive</li><li>• Hash table conflict</li></ul>

## 13.2.17 Workload Manager

### 13.2.17.1 WLM\_CGROUP\_CONFIG

**WLM\_CGROUP\_CONFIG** displays information about a Cgroup for a job that is being executed.

**Table 13-182** WLM\_CGROUP\_CONFIG columns

Name	Type	Description
cgoup_name	text	Cgroup name
priority	integer	Job priority.
usage_pencent	integer	Percentage of resources used by the Cgroup.
shares	bigint	CPU quota allocated to the Cgroup
cpuacct	bigint	Allocated CPU quota
cpuset	text	Allocated CPU cores
relpath	text	Relative path of the Cgroup
valid	text	Whether the Cgroup is valid

### 13.2.17.2 WLM\_CLUSTER\_RESOURCE\_RUNTIME

**WLM\_CLUSTER\_RESOURCE\_RUNTIME** displays a DN resource summary.

**Table 13-183** WLM\_CLUSTER\_RESOURCE\_RUNTIME columns

Name	Type	Description
min_mem_util	integer	Minimum memory usage of a DN
max_mem_util	integer	Maximum memory usage of a DN
min_cpu_util	integer	Minimum CPU usage of a DN
max_cpu_util	integer	Maximum CPU usage of a DN
min_io_util	integer	Minimum I/O usage of a DN
max_io_util	integer	Maximum I/O usage of a DN
used_mem_rate	integer	Maximum memory usage of a physical node

### 13.2.17.3 WLM\_CONTROLGROUP\_CONFIG

**WLM\_CONTROLGROUP\_CONFIG** displays information about all Cgroups in the current database.

**Table 13-184** WLM\_CONTROLGROUP\_CONFIG columns

Name	Type	Description
name	text	Cgroup name
type	text	Cgroup type
gid	bigint	Cgroup ID
classgid	bigint	ID of the Class Cgroup to which a Workload Cgroup belongs
class	text	Class Cgroup
workload	text	Workload Cgroup
shares	bigint	CPU quota allocated to a Cgroup
limits	bigint	Limit of CPUs allocated to a Cgroup
wdlevel	bigint	Workload Cgroup level
cpucores	text	Usage of CPU cores in a Cgroup

### 13.2.17.4 WLM\_RESOURCEPOOL\_RUNTIME

**WLM\_RESOURCEPOOL\_RUNTIME** displays statistics about a resource pool.

**Table 13-185** WLM\_RESOURCEPOOL\_RUNTIME columns

Name	Type	Description
rpoid	oid	OID of the resource pool
respool	name	Name of the resource pool
control_group	name	Cgroup associated with the resource pool
parentid	oid	OID of the parent resource pool
ref_count	integer	Number of jobs associated with the resource pool
active_points	integer	Number of used points in the resource pool
running_count	integer	Number of jobs running in the resource pool
waiting_count	integer	Number of jobs queuing in the resource pool
io_limits	integer	IOPS upper limit of the resource pool
io_priority	integer	I/O priority of the resource pool

### 13.2.17.5 WLM\_USER\_RESOURCE\_CONFIG

**WLM\_USER\_RESOURCE\_CONFIG** displays the resource configuration information of a user.

**Table 13-186** WLM\_USER\_RESOURCE\_CONFIG columns

Name	Type	Description
userid	oid	OID of the user
username	name	Username
sysadmin	boolean	Whether the user has the <b>sysadmin</b> permission
rpoid	oid	OID of the resource pool
respool	name	Name of the resource pool
parentid	oid	OID of the parent user
totalspace	bigint	Size of the occupied space
spacelimit	bigint	Upper limit of the space size
childcount	integer	Number of child users

Name	Type	Description
childlist	texto	Child user list

### 13.2.17.6 WLM\_USER\_RESOURCE\_RUNTIME

**WLM\_USER\_RESOURCE\_RUNTIME** displays resource usage of all users. Only administrators can query this view. This view is valid only when the GUC parameter **use\_workload\_manager** is set to **on**.

**Table 13-187** WLM\_USER\_RESOURCE\_RUNTIME columns

Name	Type	Description
username	name	Username
used_memory	integer	Size of the memory being used (unit: MB)
total_memory	integer	Available memory (unit: MB) The value <b>0</b> indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	integer	Number of CPU cores in use
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node
used_space	bigint	Used storage space (unit: KB)
total_space	bigint	Available storage space (unit: KB). The value <b>-1</b> indicates that the space is not limited.
used_temp_space	bigint	Used temporary space (reserved column; unit: KB)
total_temp_space	bigint	Available temporary space (reserved column; unit: KB). The value <b>-1</b> indicates that the maximum temporary storage space is not limited.
used_spill_space	bigint	Used space for storing spilled data (reserved column; unit: KB)
total_spill_space	bigint	Available space for storing spilled data (reserved column; unit: KB). The value <b>-1</b> indicates that the maximum space for storing spilled data is not limited.

### 13.2.17.7 WLM\_WORKLOAD\_RUNTIME

WLM\_WORKLOAD\_RUNTIME displays the status of jobs being executed by the current user on the current CN.

**Table 13-188** WLM\_WORKLOAD\_RUNTIME columns

Name	Type	Description
node_name	text	Name of the CN where the job is executed.
thread_id	bigint	Backend thread ID.
processid	integer	PID of the backend thread.
time_stamp	bigint	Time when the statement execution starts.
username	name	Name of the user logged in to the backend.
memory	integer	Memory required by the statement.
active_points	integer	Number of resources consumed by the statement in the resource pool.
max_points	integer	Number of resources consumed by the statement in the resource pool.
priority	integer	Job priority.
resource_pool	text	Resource pool to which the job belongs.
status	text	Job execution status. Its value can be: <ul style="list-style-type: none"><li>● <b>pending</b>: blocked status</li><li>● <b>running</b>: running status</li><li>● <b>finished</b>: final status</li><li>● <b>aborted</b>: termination status</li><li>● <b>unknown</b>: unknown status</li></ul>
control_group	text	Cgroups used by the job
enqueue	text	Queue that the job is in. Its value can be: <ul style="list-style-type: none"><li>● <b>GLOBAL</b>: global queue</li><li>● <b>RESPOOL</b>: resource pool queue</li><li>● <b>ACTIVE</b>: not in a queue</li></ul>
query	text	Statement being executed.

### 13.2.17.8 GLOBAL\_WLM\_WORKLOAD\_RUNTIME

**GLOBAL\_WAL\_WORKLOAD\_RUNTIME** displays the status of jobs executed by the current user on CNs. This view is accessible only to users with system administrator permissions.

**Table 13-189** GLOBAL\_WAL\_WORKLOAD\_RUNTIME columns

Name	Type	Description
node_name	text	Name of the CN where the job is executed
thread_id	bigint	Backend thread ID.
processid	integer	LWP ID of the thread
time_stamp	bigint	Time when the statement starts to be executed.
username	name	Name of the user logged in to the backend.
memory	integer	Memory required by the statement
active_points	integer	Number of resources consumed by the statement in the resource pool
max_points	integer	Maximum number of resources in the resource pool
priority	integer	Job priority
resource_pool	text	Resource pool to which the job belongs
status	text	Job execution status. Its value can be: <ul style="list-style-type: none"> <li>● <b>pending</b>: blocked status</li> <li>● <b>running</b>: running status</li> <li>● <b>finished</b>: final status</li> <li>● <b>aborted</b>: termination status</li> <li>● <b>unknown</b>: unknown status</li> </ul>
control_group	name	Cgroups used by the job
enqueue	text	Queue that the job is in. Its value can be: <ul style="list-style-type: none"> <li>● <b>GLOBAL</b>: global queue</li> <li>● <b>RESPOOL</b>: resource pool queue</li> <li>● <b>ACTIVE</b>: not in a queue</li> </ul>
query	text	Statement being executed

### 13.2.17.9 LOCAL\_IO\_WAIT\_INFO

Returns the real-time statistics of I/O control on the current node.

**Table 13-190** LOCAL\_IO\_WAIT\_INFO columns

Name	Type	Description
node_name	text	Node name.
device_name	text	Name of the data disk mounted to the node.
read_per_second	float	Number of read completions per second.
write_per_second	float	Number of write completions per second.
write_ratio	float	Ratio of the disk write I/Os to the total I/Os.
io_util	float	Percentage of the I/O time to the total CPU time per second.
total_io_util	integer	Level of the CPU time occupied by the last three I/Os. The value ranges from 0 to 6.
tick_count	integer	Interval for updating disk I/O information. The value is fixed to 1 second. The value is cleared each time before data is read.
io_wait_list_len	integer	Size of the I/O request thread wait queue. If the value is <b>0</b> , no I/O is under control.

### 13.2.17.10 GLOBAL\_IO\_WAIT\_INFO

Returns the real-time statistics of I/O control on all nodes.

**Table 13-191** GLOBAL\_IO\_WAIT\_INFO columns

Name	Type	Description
node_name	text	Node name.
device_name	text	Name of the data disk mounted to the node.
read_per_second	float	Number of read completions per second.
write_per_second	float	Number of write completions per second.
write_ratio	float	Ratio of the disk write I/Os to the total I/Os.
io_util	float	Percentage of the I/O time to the total CPU time per second.
total_io_util	integer	Level of the CPU time occupied by the last three I/Os. The value ranges from 0 to 6.

Name	Type	Description
tick_count	integer	Interval for updating disk I/O information. The value is fixed to 1 second. The value is cleared each time before data is read.
io_wait_list_len	integer	Size of the I/O request thread wait queue. If the value is <b>0</b> , no I/O is under control.

## 13.2.18 Global Plan Cache

Global plan cache (GPC) views are valid only when **enable\_global\_plancache** is set to **on**.

### 13.2.18.1 LOCAL\_PLANCACHE\_STATUS

**LOCAL\_PLANCACHE\_STATUS** displays the status of the GPC plan cache on the current node.

**Table 13-192** LOCAL\_PLANCACHE\_STATUS columns

Name	Type	Description
nodename	text	Name of the node that the plan cache belongs to
query	text	Text of query statements
refcount	integer	Number of times that the plan cache is referenced
valid	bool	Whether the plan cache is valid
databaseid	oid	ID of the database that the plan cache belongs to
schema_name	text	Schema that the plan cache belongs to
params_num	integer	Number of parameters
func_id	oid	OID of the stored procedure where the plan cache is located. If the plancache does not belong to the stored procedure, the value is <b>0</b> .
stmt_id	integer	Sequence number of the statement plan in the stored procedure.

### 13.2.18.2 GLOBAL\_PLANCACHE\_STATUS

**GLOBAL\_PLANCACHE\_STATUS** displays the status of GPC plan caches on all nodes. For details about the columns, see [LOCAL\\_PLANCACHE\\_STATUS](#).

## 13.2.19 RTO & RPO

### 13.2.19.1 global\_rto\_status

Displays log flow control information about the primary and standby nodes (except the current node and DNs).

**Table 13-193** global\_rto\_status columns

Parameter	Type	Description
node_name	text	Node name (including the primary and standby nodes)
rto_info	text	Flow control information, including the current log flow control time (unit: second) of the standby server, the expected flow control time (unit: second) specified by the GUC parameter, and the primary server sleep time (unit: $\mu$ s) required to reach the expectation

### 13.2.19.2 global\_streaming\_hadr\_rto\_and\_rpo\_stat

global\_streaming\_hadr\_rto\_and\_rpo\_stat displays the log flow control information about the primary and standby clusters for streaming DR. (This view can be used only on the CN in the primary cluster and cannot obtain statistics from the DN or standby cluster.)

**Table 13-194** Parameters

Parameter	Type	Description
hadr_sender_node_name	text	Name of the primary cluster and the main standby node in the standby cluster.
hadr_receiver_node_name	text	Name of the main standby node in the standby cluster.
current_rto	int	Flow control information, that is, log RTO time of the current primary and standby clusters (unit: second).
target_rto	int	Flow control information, that is, RTO time between the target primary and standby clusters (unit: second).
current_rpo	int	Flow control information, that is, log RPO time of the current primary and standby clusters (unit: second).
target_rpo	int	Flow control information, that is, RPO time between the target primary and standby clusters (unit: second).

Parameter	Type	Description
rto_sleep_time	int	RTO flow control information, that is, expected sleep time (unit: $\mu$ s) required by WAL sender on the host to reach the specified RTO
rpo_sleep_time	int	RPO flow control information, that is, the expected sleep time (unit: $\mu$ s) required by xlogInsert on the host to reach the specified RPO.

## 13.2.20 AI Watchdog

### 13.2.20.1 ai\_watchdog\_monitor\_status

**Table 13-195** ai\_watchdog\_monitor\_status parameters

Parameter	Type	Description
metric_name	text	Metric names: <ul style="list-style-type: none"> <li>• <b>tps</b>: TPS.</li> <li>• <b>tps_hourly</b>: average TPS per hour.</li> <li>• <b>shared_used_mem</b>: the used shared memory (MB).</li> <li>• <b>dynamic_used_shrctx</b>: the used shared memory context (MB).</li> <li>• <b>other_used_mem</b>: other used memory (MB).</li> <li>• <b>process_used_mem</b>: the used resident memory (MB).</li> <li>• <b>dynamic_used_mem</b>: the used dynamic memory (MB).</li> <li>• <b>malloc_failures</b>: the number of memory allocation failures in each collection interval.</li> <li>• <b>D_state_rate</b>: percentage of threads in the D state.</li> <li>• <b>R_state_rate</b>: percentage of threads in the R state.</li> <li>• <b>S_state_rate</b>: ratio of threads in the S state.</li> <li>• <b>db_state</b>: database state (<b>68</b> indicates D, <b>82</b> indicates R, and <b>83</b> indicates S).</li> <li>• <b>cpu_usage</b>: CPU usage. The upper limit is 100.</li> <li>• <b>disk_io</b>: disk I/O delay between two collection intervals.</li> <li>• <b>network_io</b>: network I/O delay between two collection intervals.</li> <li>• <b>threadpool_usage</b>: thread pool usage.</li> <li>• <b>threadpool_hang_rate</b>: percentage of the thread pool group in the hang state.</li> </ul>
max_length	int	Collection queue length.
current_length	int	The number of currently collected samples.
collection_interval	int	Collection interval, in seconds.
latest_value	int	Value collected last time. If no value is collected, the value is <b>null</b> .
last_report	timestamp	The last collection time.

### 13.2.20.2 ai\_watchdog\_detection\_warnings

**Table 13-196** ai\_watchdog\_detection\_warnings parameters

Parameter	Type	Description
event	text	Event name.
cause	text	Event cause.
details	text	Event details.
time	timestamp	Reporting time.
need_to_handle	bool	Determines whether automatic processing is required.

### 13.2.20.3 ai\_watchdog\_parameters

Table 13-197 ai\_watchdog\_parameters parameters

Parameter	Type	Description
name	text	Parameter name. The options are as follows: <ul style="list-style-type: none"> <li>• <b>enable_ai_watchdog</b>: determines whether to enable this function.</li> <li>• <b>ai_watchdog_max_consuming_time_ms</b>: specifies the maximum duration.</li> <li>• <b>ai_watchdog_used_memory_kb</b>: specifies the memory used by this function.</li> <li>• <b>ai_watchdog_detection_times</b>: specifies the number of detection times.</li> <li>• <b>enable_self_healing</b>: determines whether self-healing can be performed after a problem is detected.</li> <li>• <b>oom_detected_times</b>: specifies the number of detected OOMs.</li> <li>• <b>hang_detected_times</b>: specifies the number of detected hang times.</li> <li>• <b>enable_oom_detection</b>: determines whether the OOM detection function is automatically enabled.</li> <li>• <b>in_wait_time</b>: determines whether the system is waiting.</li> <li>• <b>other_used_memory_has_risk</b>: determines whether other memory usage is risky.</li> <li>• <b>shared_used_mem_has_risk</b>: determines whether risks exist when the shared memory context is used.</li> <li>• <b>dynamic_used_shrctx_has_risk</b>: determines whether the dynamic memory usage is risky.</li> </ul>
value	text	Parameter value.

## 13.2.21 Discarded

### 13.2.21.1 Query

#### 13.2.21.1.1 GS\_SLOW\_QUERY\_INFO

**GS\_SLOW\_QUERY\_INFO** displays the slow query information that has been dumped on the current node. Data is dumped from the kernel to this system catalog. If the GUC parameter **enable\_resource\_record** is set to **on**, the system

imports the query information from the kernel to **GS\_WLM\_SESSION\_QUERY\_INFO\_ALL** every 3 minutes. This operation occupies storage space and affects performance. You can check **GS\_SLOW\_QUERY\_INFO** to view the slow query information that has been dumped. This view has been discarded in this version.

**Table 13-198** GS\_SLOW\_QUERY\_INFO columns

Name	Type	Description
dbname	text	Database name
schemaname	text	Schema name
nodename	text	Node name
username	text	Username
queryid	bigint	Normalization ID
query	text	Query statement
start_time	timestamp with time zone	Execution start time
finish_time	timestamp with time zone	Execution end time
duration	bigint	Execution duration (unit: ms)
query_plan	text	Plan information
n_returned_rows	bigint	Number of rows in the result set returned by the <b>SELECT</b> statement
n_tuples_fetched	bigint	Number of rows randomly scanned
n_tuples_returned	bigint	Number of rows sequentially scanned
n_tuples_inserted	bigint	Number of rows inserted
n_tuples_updated	bigint	Number of rows updated
n_tuples_deleted	bigint	Number of rows deleted
n_blocks_fetched	bigint	Number of cache loading times
n_blocks_hit	bigint	Cache hits
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: $\mu$ s)

Name	Type	Description
cpu_time	bigint	CPU time (unit: $\mu$ s)
execution_time	bigint	Execution time in the executor (unit: $\mu$ s)
parse_time	bigint	SQL parsing time (unit: $\mu$ s)
plan_time	bigint	SQL plan generation time (unit: $\mu$ s)
rewrite_time	bigint	SQL rewriting time (unit: $\mu$ s)
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: $\mu$ s)
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: $\mu$ s)
net_send_time	bigint	Network time (unit: $\mu$ s)
data_io_time	bigint	I/O time (unit: $\mu$ s)

#### 13.2.21.1.2 GS\_SLOW\_QUERY\_HISTORY

**GS\_SLOW\_QUERY\_HISTORY** displays the slow query information that is not dumped on the current node. For details, see [18.9.15 GS\\_SLOW\\_QUERY\\_INFO](#). This is discarded in the current version.

#### 13.2.21.1.3 GLOBAL\_SLOW\_QUERY\_HISTORY

**GS\_SLOW\_QUERY\_HISTORY** displays the slow query information that is not dumped on all nodes. This view is discarded in this version. For details, see [18.9.15 GS\\_SLOW\\_QUERY\\_INFO](#).

#### 13.2.21.1.4 GLOBAL\_SLOW\_QUERY\_INFO

**GS\_SLOW\_QUERY\_HISTORY** displays the slow query information that has been dumped on all nodes. This view is discarded in this version. For details, see [18.9.15 GS\\_SLOW\\_QUERY\\_INFO](#).

### 13.2.21.2 Global Plancache

#### 13.2.21.2.1 LOCAL\_PREPARE\_STATEMENT\_STATUS

**LOCAL\_PREPARE\_STATEMENT\_STATUS** displays the information about prepare statements corresponding to the GPC plan cache on the current node.

**Table 13-199** LOCAL\_PREPARE\_STATEMENT\_STATUS columns

Name	Type	Description
nodename	text	Name of the node that the statement belongs to
cn_sess_id	bigint	Session ID of the CN that the statement is sent from
cn_node_id	integer	Node ID of the CN that the statement is sent from
cn_time_line	integer	Number of restart times of the CN that the statement is sent from
statement_name	text	Statement name
refcount	integer	Number of times that the corresponding plan cache is referenced
is_shared	bool	Whether the corresponding plan cache is shared
query	text	Corresponding query statement.

### 13.2.21.2 GLOBAL\_PREPARE\_STATEMENT\_STATUS

**GLOBAL\_PREPARE\_STATEMENT\_STATUS** displays the information about prepare statements corresponding to GPC plan caches on all nodes. For details about the columns, see [LOCAL\\_PREPARE\\_STATEMENT\\_STATUS](#).

## 13.3 DBE\_SQL\_UTIL Schema

The DBE\_SQL\_UTIL schema stores tools for managing SQL patches, including creating, deleting, enabling, and disabling SQL patches. Common users have only the USAGE permission and do not have the CREATE, ALTER, DROP, and COMMENT permissions.

For details about how to use the DBE\_SQL\_UTIL schema, see [Tuning with SQL PATCH](#).

### 13.3.1 DBE\_SQL\_UTIL.create\_hint\_sql\_patch

**create\_hint\_sql\_patch** creates hint SQL patches on the connected CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

#### NOTE

- SQL patches are not shared among CNs and need to be created on each CN.
- If load balancing is enabled or a CN needs to be specified, you are advised to use the [DBE\\_SQL\\_UTIL.create\\_remote\\_hint\\_sql\\_patch](#) API to create a CN.

**Table 13-200** DBE\_SQL\_UTIL.create\_hint\_sql\_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique query ID.
hint_string	IN text	Hint text.
description	IN text	Patch description. The default value is <b>NULL</b> .
enabled	IN bool	Specifies whether the patch takes effect. The default value is <b>true</b> .
result	OUT bool	Specifies whether this operation is successful.

### 13.3.2 DBE\_SQL\_UTIL.create\_abort\_sql\_patch

**create\_abort\_sql\_patch** creates abort SQL patches on the connected CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

 **NOTE**

- SQL patches are not shared among CNs and need to be created on each CN.
- If load balancing is enabled or a CN needs to be specified, you are advised to use the [DBE\\_SQL\\_UTIL.create\\_remote\\_abort\\_sql\\_patch](#) API to create a CN.

**Table 13-201** DBE\_SQL\_UTIL.create\_abort\_sql\_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique query ID.
description	IN text	Patch description. The default value is <b>NULL</b> .
enabled	IN bool	Specifies whether the patch takes effect. The default value is <b>true</b> .
result	OUT bool	Specifies whether this operation is successful.

### 13.3.3 DBE\_SQL\_UTIL.drop\_sql\_patch

**drop\_sql\_patch** deletes SQL patches from the connected CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

 NOTE

- SQL patches are not shared among CNs and need to be executed on each CN.
- If load balancing is enabled or a CN needs to be specified, you are advised to use the [DBE\\_SQL\\_UTIL.drop\\_remote\\_sql\\_patch](#) API to delete a CN.

**Table 13-202** DBE\_SQL\_UTIL.drop\_sql\_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

### 13.3.4 DBE\_SQL\_UTIL.enable\_sql\_patch

**enable\_sql\_patch** enables SQL patches on the connected CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

 NOTE

- SQL patches are not shared among CNs and need to be executed on each CN.
- If load balancing is enabled or a CN needs to be specified, you are advised to use the [DBE\\_SQL\\_UTIL.enable\\_remote\\_sql\\_patch](#) API to enable SQL patches.

**Table 13-203** DBE\_SQL\_UTIL.enable\_sql\_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

### 13.3.5 DBE\_SQL\_UTIL.disable\_sql\_patch

**disable\_sql\_patch** disables SQL patches on the connected CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

 NOTE

- SQL patches are not shared among CNs and need to be executed on each CN.
- If load balancing is enabled or a CN needs to be specified, you are advised to use the [DBE\\_SQL\\_UTIL.disable\\_remote\\_sql\\_patch](#) API to disable SQL patches.

**Table 13-204** DBE\_SQL\_UTIL.disable\_sql\_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

### 13.3.6 DBE\_SQL\_UTIL.show\_sql\_patch

**show\_sql\_patch** displays the SQL patch corresponding to a specified patch name and returns the running result.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

**Table 13-205** DBE\_SQL\_UTIL.show\_sql\_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	OUT bigint	Global unique query ID.
enabled	OUT bool	Determines whether the patch takes effect.
abort	OUT bool	Determines whether the value is <b>AbortHint</b> .
hint_str	OUT text	Hint text.

### 13.3.7 DBE\_SQL\_UTIL.create\_hint\_sql\_patch

**create\_hint\_sql\_patch** creates hint SQL patches and returns whether the execution is successful. This function is an overloaded function of the original function. The value of **parent\_unique\_sql\_id** can be used to limit the effective range of the hint patch.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

**Table 13-206** DBE\_SQL\_UTIL.create\_hint\_sql\_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique query ID.
parent_unique_sql_id	IN bigint	Global unique ID of the outer SQL statement. The value <b>0</b> indicates that the SQL patch statement outside the stored procedure is restricted to take effect. A non-zero value indicates that the specific stored procedure is restricted to take effect.
hint_string	IN text	Hint text.
description	IN text	Patch description. The default value is <b>NULL</b> .
enabled	IN bool	Specifies whether the patch takes effect. The default value is <b>true</b> .
result	OUT bool	Specifies whether this operation is successful.

### 13.3.8 DBE\_SQL\_UTIL.create\_abort\_sql\_patch

**create\_abort\_sql\_patch** creates abort SQL patches and returns whether the execution is successful. This function is an overloaded function of the original function. The value of **parent\_unique\_sql\_id** can be used to limit the effective range of the abort patch.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

**Table 13-207** DBE\_SQL\_UTIL.create\_abort\_sql\_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique query ID.
parent_unique_sql_id	IN bigint	Global unique ID of the outer SQL statement. The value <b>0</b> indicates that the SQL patch statement outside the stored procedure is restricted to take effect. A non-zero value indicates that the specific stored procedure is restricted to take effect.
description	IN text	Patch description. The default value is <b>NULL</b> .
enabled	IN bool	Specifies whether the patch takes effect. The default value is <b>true</b> .

Parameter	Type	Description
result	OUT bool	Specifies whether this operation is successful.

### 13.3.9 DBE\_SQL\_UTIL.create\_remote\_hint\_sql\_patch

**create\_remote\_hint\_sql\_patch** creates hint SQL patches on a specified CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

**Table 13-208** DBE\_SQL\_UTIL.create\_remote\_hint\_sql\_patch input parameters and return values

Parameter	Type	Description
node_name	IN text	Destination node name.
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique query ID.
hint_string	IN text	Hint text.
description	IN text	Patch description. The default value is <b>NULL</b> .
enabled	IN bool	Specifies whether the patch takes effect. The default value is <b>true</b> .
result	OUT bool	Specifies whether this operation is successful.

### 13.3.10 DBE\_SQL\_UTIL.create\_remote\_abort\_sql\_patch

**create\_abort\_sql\_patch** creates abort SQL patches and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

**Table 13-209** DBE\_SQL\_UTIL.create\_remote\_abort\_sql\_patch input parameters and return values

Parameter	Type	Description
node_name	IN text	Destination node name.
patch_name	IN name	Patch name.

Parameter	Type	Description
unique_sql_id	IN bigint	Global unique query ID.
description	IN text	Patch description. The default value is <b>NULL</b> .
enabled	IN bool	Specifies whether the patch takes effect. The default value is <b>true</b> .
result	OUT bool	Specifies whether this operation is successful.

### 13.3.11 DBE\_SQL\_UTIL.drop\_remote\_sql\_patch

**drop\_sql\_patch** deletes SQL patches from a specified CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

**Table 13-210** DBE\_SQL\_UTIL.drop\_remote\_sql\_patch input parameters and return values

Parameter	Type	Description
node_name	IN text	Destination node name.
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

### 13.3.12 DBE\_SQL\_UTIL.enable\_remote\_sql\_patch

**enable\_remote\_sql\_patch** enables SQL patches on a specified CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

**Table 13-211** DBE\_SQL\_UTIL.enable\_remote\_sql\_patch input parameters and return values

Parameter	Type	Description
node_name	IN text	Destination node name.
patch_name	IN name	Patch name.

Parameter	Type	Description
result	OUT bool	Specifies whether this operation is successful.

### 13.3.13 DBE\_SQL\_UTIL.disable\_remote\_sql\_patch

**disable\_remote\_sql\_patch** disables SQL patches on a specified CN and returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

**Table 13-212** DBE\_SQL\_UTIL.disable\_remote\_sql\_patch input parameters and return values

Parameter	Type	Description
node_name	IN text	Destination node name.
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

# 14 Configuring Running Parameters

## 14.1 Viewing Parameters

GaussDB uses a set of default running parameters after it is installed. You can modify the GUC parameters to enable GaussDB to better fit your service scenarios and data volume.

### Procedure

**Step 1** Connect to the database.

**Step 2** View the parameter values in the database.

- Method 1: Run the **SHOW** command.
  - Run the following command to view the value of a certain parameter:  
`gaussdb=# SHOW server_version;`  
*server\_version* indicates the database version.
  - Run the following command to view values of all parameters:  
`gaussdb=# SHOW ALL;`
- Method 2: Query the **pg\_settings** view.
  - Run the following command to view the value of a certain parameter:  
`gaussdb=# SELECT * FROM pg_settings WHERE NAME='server_version';`
  - Run the following command to view values of all parameters:  
`gaussdb=# SELECT * FROM pg_settings;`

----End

### Example

Check the character encoding type of the client.

```
gaussdb=# SHOW client_encoding;
client_encoding

UTF8
(1 row)
```

## 14.2 Setting Parameters

You are advised to modify some parameters on the GaussDB console. If the parameters cannot be modified on the console, evaluate the risks and contact customer service.

### Background

GaussDB provides multiple methods to set GUC parameters for databases, users, or sessions.

- Parameter names are case-insensitive.
- The parameter values can be integers, floating points, strings, Boolean values, or enumerated values.
  - The Boolean values can be **on / off, true / false, yes / no**, or **1 / 0**, and are case-insensitive.
  - The enumerated value range is specified in the **enumvals** column of the system catalog **pg\_settings**.
- For parameters using units, specify their units during the setting. Otherwise, default units are used.
  - The default units are specified in the **unit** column of **pg\_settings**.
  - The unit of memory can be KB, MB, or GB.
  - The unit of time can be ms, s, min, h, or d.
- You can set parameters related to CNs and DNPs at a time, but cannot do the same to other parameters.

For details about the parameters, see [GUC Parameters](#).

### Setting GUC Parameters

GaussDB provides six types of GUC parameters. For details about parameter types and their setting methods, see [Table 14-1](#).

**Table 14-1** GUC parameter types

Parameter Type	Description	Setting Method
INTERNAL	Fixed parameter. It is set during database creation and cannot be modified. Users can only view the parameter by running the <b>SHOW</b> command or in the <b>pg_settings</b> view.	None
POSTMASTER	Database server parameter. It can be set when the database is started or in the configuration file.	Method 1 in <a href="#">Table 14-2</a> .

Parameter Type	Description	Setting Method
SIGHUP	Global database parameter. It can be set when the database is started or be modified later.	Method 1 or 2 in <a href="#">Table 14-2</a> .
BACKEND	Session connection parameter. It is specified during session connection creation and cannot be modified after that. The parameter setting becomes invalid when the session is disconnected. This is an internal parameter and not recommended for users to set it.	Method 1 or 2 in <a href="#">Table 14-2</a> . <b>NOTE</b> The parameter setting takes effect when the next session is created.
SUSET	Database administrator parameter. It can be set by common users when or after the database is started. It can also be set by database administrators using SQL statements.	Method 1 or 2 by a common user, or method 3 by a database administrator in <a href="#">Table 14-2</a> .
USERSET	Common user parameter. It can be set by any user at any time.	Method 1, 2, or 3 in <a href="#">Table 14-2</a> . <b>NOTE</b> When you set parameters of the USERSET type, the parameter value set using <b>ALTER DATABASE</b> takes precedence over that set using <b>gs_guc</b> . To make the parameter settings of <b>gs_guc</b> take effect, run the <b>alter database xxx reset xxx</b> command to reset the parameters.

You can set GUC parameters in GaussDB using the methods listed in [Table 14-2](#).

**Table 14-2** Methods for setting GUC parameters

No.	Setting Method
Method 1	<ol style="list-style-type: none"> <li>1. Log in to the management console.</li> <li>2. On the <b>Instances</b> page, click the name of the target instance to go to the <b>Basic Information</b> page.</li> <li>3. In the navigation pane on the left, click <b>Parameters</b>. On the displayed page, modify parameters. If the parameters cannot be modified on the console, evaluate the modification risks and contact customer service for modification.</li> <li>4. Reboot the instance to make the modifications take effect.</li> </ol> <p><b>NOTE</b> Rebooting instances will interrupt user operations. Plan a proper execution window before the reboot.</p>
Method 2	<ol style="list-style-type: none"> <li>1. Log in to the management console.</li> <li>2. On the <b>Instances</b> page, click the name of the target instance to go to the <b>Basic Information</b> page.</li> <li>3. In the navigation pane on the left, click <b>Parameters</b>. On the displayed page, modify parameters. If the parameters cannot be modified on the console, evaluate the modification risks and contact customer service for modification.</li> </ol>
Method 3	<p>Modify a session-level parameter.</p> <ul style="list-style-type: none"> <li>• Set a session-level parameter. <code>openGauss=# SET <i>paraname</i> TO <i>value</i>;</code></li> </ul> <p>Parameter value in the current session is changed. After you exit the session, the setting becomes invalid.</p>

 **CAUTION**

If you use method 1 or 2 to set a parameter that does not belong to the current environment, the database displays a message indicating that the parameter is not supported.

If the parameters cannot be modified using the preceding methods, evaluate the risks and contact customer service for modification.

## 14.3 GUC Parameters

### 14.3.1 GUC Parameter Usage

A database provides many operation parameters. Configurations of these parameters affect the behavior of the database system. Before modifying these parameters, learn the impact of these parameters on the database. Otherwise, unexpected results may occur.

You are advised to modify some parameters on the GaussDB console. If the parameters cannot be modified on the console, evaluate the risks and contact customer service.

## Precautions

- If the value range of a parameter is a string, the string should comply with the naming conventions of the path and file name in the OS running the target database.
- If the maximum value of a parameter is *INT\_MAX*, the maximum parameter value varies by OS. *INT\_MAX* indicates the maximum value of the INT data type. The value is **2147483647**.
- If the maximum value of a parameter is *DBL\_MAX*, the maximum parameter value varies by OS. *DBL\_MAX* indicates the maximum value of the FLOAT data type.

## 14.3.2 File Location

After a database has been installed, three configuration files (**postgresql.conf**, **pg\_hba.conf**, and **pg\_ident.conf**) are automatically generated and saved in the data directory. You can use the methods described in this section to change the names and save paths of these configuration files.

When changing the storage directory of a configuration file, set **data\_directory** in **postgresql.conf** to the actual data directory.

---

### NOTICE

If a configuration file is incorrectly modified, the database will be seriously affected. Do not modify the configuration files mentioned in this section after installation.

---

## data\_directory

**Parameter description:** Specifies the GaussDB **data** directory. Only the sysadmin user can access this parameter. You can set this parameter using one of the following methods:

- Set it when you install the GaussDB.
- This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, consisting of one or more characters.

**Default value:** Specify this parameter during installation. If this parameter is not specified during installation, the database is not initialized by default.

## config\_file

**Parameter description:** Specifies the configuration file (**postgresql.conf**) of the primary server.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, consisting of one or more characters.

**Default value:** `postgresql.conf` (The absolute directory of this file may be displayed in the actual situation.)

## hba\_file

**Parameter description:** Specifies the configuration file (`pg_hba.conf`) for host-based authentication (HBA). This parameter can be specified only in the `postgresql.conf` file and can be accessed only by the `sysadmin` user.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** `pg_hba.conf` (The absolute directory of this file may be displayed in the actual situation.)

## ident\_file

**Parameter description:** Specifies the name of the configuration file (`pg_ident.conf`) for client authentication. Only the `sysadmin` user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** `pg_ident.conf` (The absolute directory of this file may be displayed in the actual situation.)

## external\_pid\_file

**Parameter description:** Specifies the extra PID file that can be used by the server management program. Only the `sysadmin` user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

This parameter takes effect only after the database restarts.

---

**Value range:** a string

**Default value:** empty

## enable\_default\_cfunc\_libpath

**Parameter description:** Specifies GaussDB whether the `.so` file uses the default path when the C function is created.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**on:** indicates that the .so file must be placed in the specified directory (\$libdir/proc\_srclib) when the C function is created.

**off:** indicates that the .so file can be stored in any accessible directory when the C function is created.

**Default value:** on

---

**NOTICE**

If this parameter is set to **off**, the .so file can be placed in any accessible directory or the .so file provided by the system can be used, which poses security risks. Therefore, you are not advised to set this parameter to **off**.

---

## 14.3.3 Connection and Authentication

### 14.3.3.1 Connection Settings

This section describes parameters related to client-server connection modes.

#### light\_comm

**Parameter description:** Specifies whether the server uses the lightweight communication mode.

This parameter specifies whether the server uses the communication mode based on lightweight locks and non-blocking sockets. This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **TRUE / ON:** indicates that the lightweight communication mode is used.
- **FALSE / OFF:** indicates that the lightweight communication mode is not used.

**Default value:** FALSE / OFF

#### listen\_addresses

**Parameter description:** Specifies the TCP/IP address of the client for a server to listen on.

This parameter specifies the IP address used by the GaussDB server for listening, for example, IPv4. Multiple NICs may exist on the host and each NIC can be bound to multiple IP addresses. This parameter specifies the IP addresses to which GaussDB is bound. The client can use the IP address specified by this parameter to connect to GaussDB or send requests to GaussDB.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:**

- Host name or IP address. Multiple values are separated with commas (,).
- Asterisk (\*) or **0.0.0.0**, indicating that all IP addresses will be listened on, which is not recommended due to potential security risks.
- If the parameter is not specified, the server does not listen on any IP address. In this case, only Unix domain sockets can be used for database connections.

**Default value:**

After the cluster is installed, configure different default values based on the IP addresses of different instances in the **public\_cloud.conf** file. The default value of CN is **listen\_addresses = 'localhost,IP address of the mgr.net NIC,IP address of the data.net NIC,IP address of the virtual.net NIC'**. The default value of DN is **listen\_addresses = 'IP address of the data.net NIC'**.

 **NOTE**

**localhost** indicates that only local loopback is allowed.

The **public\_cloud.conf** file contains the following NIC information: **mgr.net** (management NIC), **data.net** (data NIC), and **virtual.net** (virtual NIC).

## listen\_address\_ext

**Parameter description:** Specifies the extended TCP/IP address of the client for a server to listen on. This parameter specifies that the GaussDB server uses a specific IP address as the extended listening IP address.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an IPv4 address. Multiple IP addresses are not supported.

**Default value:** "localhost"

 **NOTE**

- The setting of the **listen\_address\_ext** parameter involves the network topology and configuration of the extended IP addresses of the entire cluster. Therefore, it must be set and reloaded by using the OM tool.
- **listen\_address\_ext** is a network channel used to extend specific IP addresses. If this parameter is set to \*, **0.0.0.0**, **localhost**, or **127.0.0.1**, the setting does not take effect. If required, you are advised to set these addresses in **listen\_addresses**.
- **listen\_address\_ext** is used to configure distributed DNs. It does not take effect if it is configured for CNs.

## local\_bind\_address

**Parameter description:** Specifies the host IP address bound to the current node for connecting to other nodes in the cluster.

This parameter is a POSTMASTER parameter.

 NOTE

This parameter is specified in the configuration file during installation. Do not modify this parameter unless absolutely necessary. Otherwise, database communication will be affected.

**Default value:**

After the cluster is installed, configure different default values based on the IP addresses of different instances in the **public\_cloud.conf** file. The default value of CN/DN is **local\_bind\_address = 'IP address of the data.net NIC'**.

 NOTE

The **public\_cloud.conf** file contains the following NIC information: **mgr.net** (management NIC), **data.net** (data NIC), and **virtual.net** (virtual NIC).

## port

**Parameter description:** Specifies the TCP port listened on by the GaussDB.

 NOTE

This parameter is specified in the configuration file during installation. Do not modify this parameter unless absolutely necessary. Otherwise, database communication will be affected.

**Value range:** an integer ranging from 1 to 65535.

 NOTE

- When setting the port number, ensure that the port number is not in use. When setting the port numbers of multiple instances, ensure that the port numbers do not conflict.
- Ports 1 to 1023 are reserved for the operating system. Do not use them.
- When the cluster is installed using the configuration file, pay attention to the ports reserved in the communication matrix in the configuration file. For example, the port specified by the value of **dataPortBase** plus 1 needs to be reserved for internal tools, and the port specified by the value of **dataPortBase** plus 6 needs to be reserved as the communication port of the flow engine message queue. (Due to specification changes, the current version no longer supports this feature. Do not use it.) Therefore, during cluster installation, the maximum value of **port** is **65532** for CNs, **65529** for DN, and **65534** for GTMs. Ensure that the port number does not conflict with each other.

**Default value:** **5432** (The actual value is specified in the configuration file during installation.)

## max\_connections

**Parameter description:** Specifies the maximum number of concurrent connections to the database. This parameter influences the concurrent processing capability of the cluster.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer. The minimum value is **10** (greater than **max\_wal\_senders**). The theoretical maximum value is **262143**. The actual maximum value is a dynamic value, which is calculated using the formula:  $262143$

– **job\_queue\_processes** – **autovacuum\_max\_workers** – **max\_inner\_tool\_connections** – **max\_concurrent\_autonomous\_transactions** – **AUXILIARY\_BACKENDS** – **AV\_LAUNCHER\_PROCS** – **min(max(newValue/4,64),1024)**. The values of **job\_queue\_processes**, **autovacuum\_max\_workers**, **max\_inner\_tool\_connections**, and **max\_concurrent\_autonomous\_transactions** depend on the settings of the corresponding GUC parameters. **AUXILIARY\_BACKENDS** indicates the number of reserved auxiliary threads, which is fixed to **20**. **AV\_LAUNCHER\_PROCS** indicates the number of reserved autovacuum launcher threads, which is fixed to **2**. In **min(max(newValue/4,64),1024)**, **newValue** indicates the new value.

For different memory specifications, the value range of this parameter in different instances is as follows:

**Table 14-3** Value ranges of memory specifications for different instances in independent deployment mode

Memory Specifications	Value Range for CNs	Value Range for DNs
< 32 GB	[10,100]	[10,100]
[32 GB,64 GB)	[10,200]	[10,200]
[64 GB,128 GB)	[10,1000]	[10,2500]
[128 GB,256 GB)	[10,2000]	[10,6000]
[256 GB,480 GB)	[10,4000]	[10,12000]
≥ 480 GB	[10,8000]	[10,24000]

**Default value:**

- Independent deployment:  
 CN: **8000** (60-core CPU/480 GB memory); **4000** (32-core CPU/256 GB memory); **2000** (16-core CPU/128 GB memory); **1000** (8-core CPU/64 GB memory); **100** (4-core CPU/32 GB memory and 4-core CPU/16 GB memory)  
 DN: **24000** (60-core CPU/480 GB memory); **12000** (32-core CPU/256 GB memory); **6000** (16-core CPU/128 GB memory); **2500** (8-core CPU/64 GB memory); **100** (4-core CPU/32 GB memory and 4-core CPU/16 GB memory)

**Impact of incorrect configuration:**

If the value of **max\_connections** exceeds the maximum dynamic value, the node fails to be started and the following error message is displayed: "invalid value for parameter "max\_connections"". Alternatively, the memory fails to be allocated during the node startup and the following error message is displayed: "Cannot allocate memory."

If only the value of **max\_connections** is increased but the memory parameter is not adjusted in proportion based on the external egress specifications, when the service pressure is high, the memory may be insufficient, and the error message "memory is temporarily unavailable" is displayed.

 NOTE

- If the number of connections of the administrator exceeds the value of **max\_connections**, the administrator can still connect to the database after the connections are used up by common users. If the number of connections exceeds the value of **sysadmin\_reserved\_connections**, an error is reported. That is, the maximum number of connections of the administrator is equal to the value of *max\_connections* + *sysadmin\_reserved\_connections*.
- For common users, internal jobs use some connections. Therefore, the value of this parameter is slightly less than that of **max\_connections**. The value depends on the number of internal connections.
- After the thread pool is enabled, the maximum number of stream threads is determined by the value of **max\_connections**. If the number of stream threads reaches the upper limit, the error is reported: "Exceed stream thread pool limitation...". In this case, you can increase the value of **max\_connections** to increase the upper limit. This parameter is a POSTMASTER parameter. Therefore, you can estimate the number of stream threads based on service requirements. The formula is as follows: Total number of stream threads = Number of concurrent services x Number of stream threads consumed by each concurrently executed statement (which can be viewed in the execution plan).

## max\_inner\_tool\_connections

**Parameter description:** Specifies the maximum number of concurrent connections of a tool which is allowed to connect to the database. This parameter influences the concurrent connection capability of the GaussDB tool.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to *MIN* (which takes the smaller value between **262143** and *max\_connections*). For details about how to calculate the value of *max\_connections*, see the preceding description.

**Default value:** 50 If the default value is greater than the maximum value supported by the kernel (determined when the **gs\_initdb** command is executed), an error message is displayed.

**Setting suggestions:**

You are advised to use the default value for this parameter in the primary database node.

## sysadmin\_reserved\_connections

**Parameter description:** Specifies the minimum number of connections reserved for administrators. You are advised not to set this parameter to a large value. This parameter is used together with the *max\_connections* parameter. The maximum number of connections of the administrator is equal to the value of *max\_connections* + *sysadmin\_reserved\_connections*.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to *MIN(262143, max\_connections)* (the smaller value between **262143** and **max\_connections**). For details about how to calculate the value of **max\_connections**, see [max\\_connections](#).

**Default value:** 3

**Note:** When the thread pool function is enabled, if the thread pool is fully occupied, a processing bottleneck occurs. As a result, connections reserved by the administrator cannot be established. In this case, you can use `gsql` to establish connections through the port specified by the primary port number plus 1 and clear useless sessions.

## service\_reserved\_connections

**Parameter description:** Specifies the minimum number of connections reserved for background O&M users (with the persistence attribute). A large value is not recommended. This parameter is used together with **max\_connections**. The maximum number of connections of an O&M user can be calculated as follows: **max\_connections + service\_reserved\_connections**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 262143.

**Default value:** 10

**Note:** If this parameter is set to a small value, the O&M user (with the persistence attribute) cannot connect to the database and jobs cannot be executed when the number of connections reaches the maximum (**max\_connections**).

## extip\_reserved\_connections

**Parameter description:** Provides a reserved connection channel for common `gsql` service connections from the extended IP address specified by **listen\_address\_ext**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to  $MIN(262143, max\_connections)$  (the smaller value between **262143** and **max\_connections**). For details about how to calculate the value of **max\_connections**, see [max\\_connections](#).

**Default value:** 5

---

### CAUTION

This parameter is used for common `gsql` service connections only when **listen\_address\_ext** is set to a specific IP address for a distributed DN. This parameter specifies the maximum number of connections allowed after the maximum number of connections specified by **max\_connections** is reached.

---

## unix\_socket\_directory

**Parameter description:** Specifies the Unix domain socket directory that the GaussDB server listens to for connections from the client. Only the **sysadmin** user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

The parameter length limit varies by OS. In the Linux OS, the length of a socket path name (combination of a socket directory and a socket file name) cannot exceed 107 bytes, and the length of a directory cannot exceed 92 bytes. If the upper limit is exceeded, the error "Unix-domain socket path xxx is too long" will be reported and the process cannot be started properly. You can retrieve the **system\_call** log in the **cm\_agent** directory to locate configuration issues.

**Value range:** a string.

**Default value:** empty. The actual value is specified by **tmpMppdbPath** in the configuration file during installation.

## unix\_socket\_group

**Parameter description:** Specifies the group of the Unix domain socket (the user of a socket is the user that starts the server). This parameter can work with [unix\\_socket\\_permissions](#) to control socket access.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. If this parameter is set to an empty string, the default group of the current user is used.

**Default value:** an empty string

## unix\_socket\_permissions

**Parameter description:** Specifies access permissions for the Unix domain socket.

The Unix domain socket uses the usual permission set of the Unix file system. The value of this parameter should be a number (acceptable for the **chmod** and **umask** commands). If a user-defined octal format is used, the number must start with 0.

You are advised to set it to **0770** (only allowing access from users connecting to the database and users in the same group as them) or **0700** (only allowing access from users connecting to the database).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0000 to 0777

**Default value:** 0700

 **NOTE**

In the Linux OS, a document has one document attribute and nine permission attributes which consist of the read (r), write (w), and execute (x) permissions of the Owner, Group, and Others groups.

The r, w, and x permissions are represented by the following numbers:

r: 4

w: 2

x: 1

-: 0

The three attributes in a group are accumulative.

For example, **-rwxrwx---** indicates the following permissions:

owner = rwx = 4+2+1 = 7

group = rwx = 4+2+1 = 7

others = --- = 0+0+0 = 0

The permission of the file is 0770.

## application\_name

**Parameter description:** Specifies the client name used in the current connection request.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

When a standby node requests to replicate logs on the primary node, if this parameter is not an empty string, it is used as the name of the streaming replication slot of the standby node on the primary node. In this case, if the length of this parameter exceeds 61 bytes, only the first 61 bytes are used as the streaming replication slot name.

**Value range:** a string. The actual query result depends on the client used for queries or user configurations.

**Default value:** an empty string

## connection\_info

**Parameter description:** Specifies the database connection information, including the driver type, driver version, driver deployment path, and process owner.

This parameter is a USERSET parameter used for O&M. You are advised not to change the parameter value.

**Value range:** a string.

**Default value:** an empty string.

 NOTE

- An empty string indicates that the driver connected to the database does not support automatic setting of the **connection\_info** parameter or the parameter is not set by users in applications.
- The following is an example of the concatenated value of **connection\_info**:  

```
{"driver_name":"ODBC","driver_version": "(GaussDB 503.1.XXX build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131 release","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"omm"}
```

By default, **driver\_name** and **driver\_version** are displayed. The display of **driver\_path** and **os\_user** is controlled by users. For details, see "Application Development Guide > Development Based on JDBC > Connecting to the Database" in *Developer Guide* and "Application Development Guide > Development Based on ODBC > Configuring a Data Source in the Linux OS" in *Developer Guide*.

## backend\_version

**Parameter description:** Specifies the version number of the synchronous connection between CNs or between a CN and a DN. This parameter involves the version number and cannot be set randomly.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 100000.

### 14.3.3.2 Security and Authentication (postgresql.conf)

This section describes parameters about client-to-server authentication.

## authentication\_timeout

**Parameter description:** Specifies the longest duration to wait before the client authentication times out. If a client is not authenticated by the server within the period, the server automatically disconnects from the client so that the client does not occupy connection resources.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 600. The smallest unit is s.

**Default value:** 1min

## auth\_iteration\_count

**Parameter description:** Specifies the number of iterations during the generation of encryption information for authentication.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 2048 to 134217728

**Default value:** 10000

---

**NOTICE**

If the number of iterations is too small, the password storage security is reduced. If the number of iterations is too large, the performance deteriorates in scenarios involving password encryption, such as authentication and user creation. Set the number of iterations based on actual hardware conditions. You are advised to retain the default value.

---

## session\_authorization

**Parameter description:** Specifies the user ID of the current session.

This is a USERSET parameter and can be set only by following the instructions provided in "SQL Reference > SQL Syntax > SET SESSION AUTHORIZATION" in *Developer Guide*.

**Value range:** a string

**Default value:** NULL

## session\_timeout

**Parameter description:** Specifies the longest duration allowed when no operations are performed on a client after it is connected to the server.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 86400. The smallest unit is s. **0** indicates that the timeout is disabled.

**Default value:** 1800

---

**NOTICE**

The gsql client of GaussDB has an automatic reconnection mechanism. For local connection of initialized users, the client reconnects to the server if the connection breaks after the timeout.

---

## ssl

**Parameter description:** Specifies whether SSL connections are enabled. Before setting this parameter, read [Using gsql to Connect to an Instance](#).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that SSL connections are enabled.
- **off** indicates that SSL connections are not enabled.

---

**NOTICE**

GaussDB supports SSL when a client connects to a CN. You are advised to enable SSL connections only on CNs. The default value is **off** on DN. Before setting this parameter to **on**, ensure that the [ssl\\_cert\\_file](#), [ssl\\_key\\_file](#), and [ssl\\_ca\\_file](#) parameters are correctly set. If SM series cryptographic algorithms are used, ensure that the [ssl\\_enc\\_cert\\_file](#) and [ssl\\_enc\\_key\\_file](#) parameters are correctly set. Incorrect settings may cause cluster startup failures.

---

**Default value:** **on** (for CNs) or **off** (for DN)

## comm\_ssl

**Parameter description:** Specifies whether to enable the SSL connection between primary DN. Before setting this parameter, read [Using gsql to Connect to an Instance](#).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the SSL connection is enabled between primary DN.
- **off** indicates that the SSL connection is disabled between primary DN.

---

**NOTICE**

- It is recommended that this parameter be enabled only on DN. The default value on CN is **off**.
  - To enable SSL connections, you also need to ensure that parameters such as [ssl\\_cert\\_file](#), [ssl\\_key\\_file](#), and [ssl\\_ca\\_file](#) are configured correctly. Incorrect configurations may cause startup failure of the cluster.
- 

**Default value:** **off**

## require\_ssl

**Parameter description:** Specifies whether the server requires SSL connections. This parameter is valid only when [ssl](#) is set to **on**. Before setting this parameter, read [Using gsql to Connect to an Instance](#).

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the server requires SSL connections.
- **off** indicates that the server does not require SSL connections.

---

**NOTICE**

GaussDB supports SSL when a client connects to a CN. It is recommended that the SSL connection be enabled only on CNs.

---

**Default value:** off

## ssl\_ciphers

**Parameter description:** Specifies the list of encryption algorithms supported by SSL. Only the sysadmin user can access the list.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. Separate multiple encryption algorithms by semicolons (;).

---

**NOTICE**

If **ssl\_ciphers** is set incorrectly, the cluster cannot be started properly.

---

**Default value:** ALL

## ssl\_renegotiation\_limit

**Parameter description:** Specifies the allowed traffic volume over an SSL-encrypted channel before the session key is renegotiated. The renegotiation mechanism reduces the probability that attackers use the password analysis method to crack the key based on a huge amount of data but causes big performance losses. The traffic indicates the sum of transmitted and received traffic. The SSL renegotiation mechanism has been disabled because of potential risks. This parameter is reserved for version compatibility and does not take effect.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is KB. **0** indicates that the renegotiation mechanism is disabled.

**Default value:** 0

## ssl\_cert\_file

**Parameter description:** Specifies the name of the file that contains the SSL server certificate. The relative path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** server.crt

## ssl\_key\_file

**Parameter description:** Specifies the name of the file that contains the SSL private key. The relative path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** server.key

## ssl\_enc\_cert\_file

**Parameter description:** Specifies the name of the SSL server certificate file that is encrypted using Chinese cryptographic algorithms. The path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** server\_enc.crt

## ssl\_enc\_key\_file

**Parameter description:** Specifies the name of the file that contains the SSL private key. The relative path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** server\_enc.key

## ssl\_ca\_file

**Parameter description:** Specifies the name of the root certificate that contains CA information. Its path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. If it is an empty string, no CA file is loaded and client certificate verification is not performed.

**Default value:** cacert.pem

## ssl\_crl\_file

**Parameter description:** Specifies the certificate revocation list (CRL). If a client certificate is in the list, the certificate is invalid. The path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. An empty string indicates that there is no CRL.

**Default value:** an empty string

## ssl\_cert\_notify\_time

**Parameter description:** Specifies the number of days prior to SSL server certificate expiration that a user will receive a reminder. When the SSL certificate is initialized during connection establishment, if the duration from the current time to the certificate expiration time is shorter than the specified value, an expiration notification is recorded in the log.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 7 to 180. The unit is day.

**Default value:** 90

## krb\_server\_keyfile

**Parameter description:** Specifies the location of the main configuration file of the Kerberos service. Only the **sysadmin** user can access the file.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** an empty string

## krb\_srvname

**Parameter description:** Specifies the Kerberos service name.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** postgres

## krb\_caseins\_users

**Parameter description:** Specifies whether the Kerberos username is case-sensitive.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the Kerberos username is case-insensitive.
- **off** indicates that the Kerberos username is case-sensitive.

**Default value:** off

## modify\_initial\_password

**Parameter description:** After GaussDB is installed, there is only one initial user account (whose UID is 10) in the database. When a user logs in to the database using this initial account for the first time, this parameter determines whether the password of the initial account needs to be modified.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

If the initial user password is not specified during the installation, the initial user password is empty by default after the installation. Before performing other operations, you need to set the initial user password using the gsql client. This parameter no longer takes effect and is reserved only for compatibility with upgrade scenarios.

---

**Value range:** Boolean

- **on** indicates that the password of the initial account needs to be modified upon the first login.
- **off** indicates that the password of the initial account does not need to be modified.

**Default value:** off

## password\_policy

**Parameter description:** Specifies whether to check the password complexity when you run the **CREATE ROLE/USER** or **ALTER ROLE/USER** command to create or modify the GaussDB account.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

For security purposes, do not disable the password complexity policy.

---

**Value range:** 0 and 1

- **0** indicates that no password complexity policy is enabled.
- **1** indicates that the default password complexity policy is enabled.

**Default value:** 1

## password\_reuse\_time

**Parameter description:** Specifies whether to check the reuse interval of the new password when you run the **ALTER USER** or **ALTER ROLE** command to change a user password.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

When you change the password, the system checks the values of [password\\_reuse\\_time](#) and [password\\_reuse\\_max](#).

- If the values of [password\\_reuse\\_time](#) and [password\\_reuse\\_max](#) are both positive numbers, an old password can be reused when it meets either of the reuse restrictions.
- If the value of [password\\_reuse\\_time](#) is **0**, password reuse is restricted based on the number of reuse times, and not on the reuse interval.
- If the value of [password\\_reuse\\_max](#) is **0**, password reuse is restricted based on the reuse interval, and not on the number of reuse times.
- If the values of both [password\\_reuse\\_time](#) and [password\\_reuse\\_max](#) are **0**, password reuse is not restricted.

---

**Value range:** a floating point number ranging from 0 to 3650. The unit is day.

- **0** indicates that the password reuse interval is not checked.
- A positive number indicates that a new password cannot be chosen from passwords in history that are newer than the specified number of days.

## password\_reuse\_max

**Parameter description:** Specifies whether to check the reuse times of the new password when you run the **ALTER USER** or **ALTER ROLE** command to change a user password. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

When you change the password, the system checks the values of [password\\_reuse\\_time](#) and [password\\_reuse\\_max](#).

- If the values of [password\\_reuse\\_time](#) and [password\\_reuse\\_max](#) are both positive numbers, an old password can be reused when it meets either of the reuse restrictions.
- If the value of [password\\_reuse\\_time](#) is **0**, password reuse is restricted based on the number of reuse times, and not on the reuse interval.
- If the value of [password\\_reuse\\_max](#) is **0**, password reuse is restricted based on the reuse interval, and not on the number of reuse times.
- If the values of both [password\\_reuse\\_time](#) and [password\\_reuse\\_max](#) are **0**, password reuse is not restricted.

---

**Value range:** an integer ranging from 0 to 1000

- **0** indicates that the password reuse times are not checked.
- A positive number indicates that the new password cannot be the one whose reuse times exceed the specified number.

**Default value:** 0

## password\_lock\_time

**Parameter description:** Specifies the duration before a locked account is automatically unlocked.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

The locking and unlocking functions take effect only when the values of **password\_lock\_time** and **failed\_login\_attempts** are positive numbers.

---

**Value range:** a floating point number ranging from 0 to 365. The unit is day. The integer part indicates the number of days, and the decimal part can be converted into hours, minutes, and seconds. For example, **password\_lock\_time=1.5** indicates one day and 12 hours.

- **0** indicates that an account is not automatically locked if the password verification fails.
- A positive number indicates the duration after which a locked account is automatically unlocked.

**Default value:** 1

## failed\_login\_attempts

**Parameter description:** Specifies the maximum number of incorrect password attempts before an account is locked. The account will be automatically unlocked after the time specified by **password\_lock\_time**. Only the **sysadmin** user can access the account. The automatic account locking policy applies in scenarios such as login and password modification using the **ALTER USER** command.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

The locking and unlocking functions take effect only when the values of **failed\_login\_attempts** and **password\_lock\_time** are positive numbers.

---

**Value range:** an integer ranging from 0 to 1000

- **0** indicates that the automatic locking function does not take effect.
- A positive number indicates that an account is locked when the number of incorrect password attempts reaches the specified number.

**Default value:** 10

## password\_encryption\_type

**Parameter description:** Specifies the encryption type of a user password. Changing the value of this parameter does not change the password encryption type of existing users. The new encryption type is applied to passwords of new users or passwords modified after the parameter value is changed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0, 1, 2, or 3

- 0 indicates that passwords are encrypted with MD5.
- 1 indicates that passwords are encrypted with SHA-256 and MD5.
- 2 indicates that passwords are encrypted with SHA-256.
- 3 indicates that the passwords are encrypted in sm3 mode.

---

### NOTICE

The MD5 encryption algorithm is not recommended because it has lower security and poses security risks.

---

**Default value:** 2

## password\_min\_length

**Parameter description:** Specifies the minimum length of an account password. Only the **sysadmin** user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer. A password can contain 6 to 999 characters.

**Default value:** 8

## password\_max\_length

**Parameter description:** Specifies the maximum length of an account password. Only the **sysadmin** user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer. A password can contain 6 to 999 characters.

**Default value:** 32

## password\_min\_uppercase

**Parameter description:** Specifies the minimum number of uppercase letters that an account password must contain. Only the **sysadmin** user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 999

- 0 means no limit.
- A positive integer indicates the minimum number of uppercase letters required in a password when you create an account.

**Default value:** 0

## password\_min\_lowercase

**Parameter description:** Specifies the minimum number of lowercase letters that an account password must contain. Only the **sysadmin** user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 999

- 0 means no limit.
- A positive integer indicates the minimum number of lowercase letters required in a password when you create an account.

**Default value:** 0

## password\_min\_digital

**Parameter description:** Specifies the minimum number of digits that an account password must contain. Only the **sysadmin** user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 999

- 0 means no limit.
- A positive integer indicates the minimum number of digits required in a password when you create an account.

**Default value:** 0

## password\_min\_special

**Parameter description:** Specifies the minimum number of special characters that an account password must contain. Only the **sysadmin** user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 999

- 0 means no limit.
- A positive integer indicates the minimum number of special characters required in a password when you create an account.

**Default value:** 0

## password\_effect\_time

**Parameter description:** Specifies the validity period of an account password.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0 to 999. The unit is day.

- 0 indicates that the validity period restriction is disabled.
- A floating-point number from 1 to 999 indicates the number of days for which an account password is valid. When the password is about to expire or has expired, the system prompts the user to change the password.

**Default value:** 0

## password\_notify\_time

**Parameter description:** Specifies how many days in advance a user is notified before a password expires.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 999. The unit is day.

- 0 indicates that the reminder is disabled.
- A positive integer indicates the number of days prior to password expiration that a user will receive a reminder.

**Default value:** 7

### 14.3.3.3 Communication Library Parameters

This section describes parameter settings and value ranges for communications libraries.

## tcp\_keepalives\_idle

**Parameter description:** Specifies the interval for transmitting keepalive signals on an OS that supports the **TCP\_KEEPIDLE** socket option. If no keepalive signal is transmitted, the connection is in idle mode.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- If the OS does not support **TCP\_KEEPIIDLE**, set this parameter to **0**.
  - The parameter is ignored on an OS where connections are established using the Unix domain socket.
- 

**Value range:** 0 to 3600. The unit is s.

**Default value:** 1min

### tcp\_keepalives\_interval

**Parameter description:** Specifies the response time before retransmission on an OS that supports the **TCP\_KEEPINTVL** socket option.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 to 180. The unit is s.

**Default value:** 30

---

**NOTICE**

- If the OS does not support **TCP\_KEEPINTVL**, set this parameter to **0**.
  - The parameter is ignored on an OS where connections are established using the Unix domain socket.
- 

### tcp\_keepalives\_count

**Parameter description:** Specifies the number of keepalive signals that can be waited before the GaussDB server is disconnected from the client on an OS that supports the **TCP\_KEEPCNT** socket option.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- If the OS does not support **TCP\_KEEPCNT**, set this parameter to **0**.
  - The parameter is ignored on an OS where connections are established using the Unix domain socket.
- 

**Value range:** 0 to 100. **0** indicates that the connection is immediately broken if GaussDB does not receive a keepalived signal from the client.

**Default value:** 20

### tcp\_user\_timeout

**Parameter description:** Specifies the maximum duration for which the transmitted data can remain in the unacknowledged state before the TCP

connection is forcibly closed when the GaussDB sends data on the OS that supports the TCP\_USER\_TIMEOUT socket option.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**NOTICE**

- If the OS does not support the TCP\_USER\_TIMEOUT option, the value of this parameter does not take effect. The default value is **0**.
- The parameter is ignored on an OS where connections are established using the Unix domain socket.

**Value Range:** 0 to 3600000. The unit is ms. The value **0** indicates that the value is set based on the OS.

**Default value:** 0

Note that the effective result of this parameter varies according to the OS kernel.

- For AArch64 EulerOS (Linux kernel version: 4.19), the timeout interval is the value of this parameter.
- For x86 Euler 2.5 (Linux kernel version: 3.10), the timeout interval is not the value of this parameter but the maximum value in different ranges. That is, the timeout interval is the maximum upper limit of the total Linux TCP retransmission duration to which the value of **tcp\_user\_timeout** belongs. For example, if **tcp\_user\_timeout** is set to **40000**, the total retransmission duration is 51 seconds.

**Table 14-4** Value of tcp\_user\_timeout for x86 Euler 2.5 (Linux kernel version: 3.10)

Number of Linux TCP Retransmission Times	Total Linux TCP Retransmission Duration Range (s)	Example of tcp_user_timeout (ms)	Actual Linux TCP Retransmission Duration (s)
1	(0.2,0.6]	400	0.6
2	(0.6,1.4]	1000	1.4
3	(1.4,3]	2000	3
4	(3,6.2]	4000	6.2
5	(6.2,12.6]	10000	12.6
6	(12.6,25.4]	20000	25.4
7	(25.4,51]	40000	51
8	(51,102.2]	80000	102.2

Number of Linux TCP Retransmission Times	Total Linux TCP Retransmission Duration Range (s)	Example of tcp_user_timeout (ms)	Actual Linux TCP Retransmission Duration (s)
9	(102.2,204.6]	150000	204.6
10	(204.6,324.6]	260000	324.6
11	(324.6,444.6]	400000	444.6

Note: The duration of each TCP retransmission increases exponentially with the number of retransmission times. When the duration of a TCP retransmission reaches 120 seconds, the duration of each subsequent retransmission does not change.

## comm\_tcp\_mode

**Parameter description:** Specifies whether the communications library uses the TCP or SCTP protocol to set up a data channel. The parameter setting takes effect after you restart the cluster. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

SCTP is no longer supported. This parameter is provided for compatibility, but its value is fixed at **on**.

---

**Value range:** Boolean. If this parameter is set to **on** for a CN, it connects to a DN using TCP. If this parameter is set to **on** for a DN, the DN communicates with each other using TCP.

**Default value:** on

## comm\_sctp\_port

**Parameter description:** Specifies the TCP or SCTP port used to listen on data packet channels by the TCP proxy communications library or SCTP communications library. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

This port number is automatically allocated during cluster deployment. Do not change the parameter. If the port number is incorrectly configured, the database communication fails.

---

**Value range:** an integer ranging from 0 to 65535

**Default value:** 25110 (The actual value is the specified value of the **port** GUC parameter plus 2.)

## comm\_control\_port

**Parameter description:** Specifies the TCP listening port used by the TCP proxy communications library or SCTP communications library. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

This port number is automatically allocated during cluster deployment. Do not change the parameter. If the port number is incorrectly configured, the database communication fails.

---

**Value range:** an integer ranging from 0 to 65535

**Default value:** 25111 (The actual value is the specified value of the **port** GUC parameter plus 3.)

## comm\_max\_datanode

**Parameter description:** Specifies the maximum number of DNs supported by the TCP proxy communications library.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 8192

**Default value:** maximum number of primary DNs supported by each node

**Recommended value:** 256

## comm\_max\_stream

**Parameter description:** Specifies the maximum number of concurrent data streams supported by the TCP proxy communications library. The value of this parameter must be greater than: Number of concurrent data streams x Number of operators in each stream x Square of smp.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 60000

**Default value:** 1024

---

**NOTICE**

- You are advised not to set this parameter to a large value because this will cause high memory usage (256 bytes x **comm\_max\_stream** x **comm\_max\_datanode**). If the number of concurrent data streams is large, the query is complex and the SMP is large, resulting in insufficient memory.
  - If the process memory is sufficient, you can properly increase the value of **comm\_max\_stream**.
- 

### comm\_max\_receiver

**Parameter description:** Specifies the maximum number of receiver threads for the TCP proxy communications library.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 50

**Default value:** 4

### comm\_quota\_size

**Parameter description:** Specifies the maximum size of packets that can be consecutively sent by the TCP proxy communications library. When you use a 1GE NIC, a small value ranging from 20 KB to 40 KB is recommended.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2048000. The default unit is KB.

**Default value:** 1 MB

### comm\_usable\_memory

**Parameter description:** Specifies the maximum memory available for buffering on the TCP proxy communications library on a DN.

---

**NOTICE**

This parameter must be set based on environment memory and the deployment method. If it is too large, an out-of-memory (OOM) exception may occur. If it is too small, the performance of the TCP proxy communications library or SCTP communications library may deteriorate. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

---

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 100 x 1024 to 1073741823. The default unit is KB.

**Default value:** 4000 MB

## comm\_memory\_pool

**Parameter description:** Specifies the size of the memory pool resources that can be used by the TCP proxy communications library on a DN.

---

### NOTICE

If the memory used by the communications library is small, set this parameter to a small value. Otherwise, set it to a large value.

---

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 100 x 1024 to 1073741823. The default unit is KB.

**Default value:** 2000 MB

## comm\_memory\_pool\_percent

**Parameter description:** Specifies the percentage of the memory pool resources that can be used by the TCP proxy communications library on a DN. This parameter is used to adaptively reserve memory used by the communications libraries.

---

### NOTICE

If the memory used by the communications library is small, set this parameter to a small value. Otherwise, set it to a large value.

---

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 100

**Default value:** 0

## comm\_client\_bind

**Parameter description:** Specifies whether to bind the client of the communications library to a specified IP address when the client initiates a connection.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the client is bound to a specified IP address.
- **off** indicates that the client is not bound to any IP addresses.

---

**NOTICE**

If multiple IP addresses of a node in the cluster are on the same network segment, set this parameter to **on**. In this case, the client is bound to the IP address specified by **listen\_addresses**. The concurrency performance of the cluster depends on the number of random ports because a port can be used by only one client at a time.

---

**Default value:** off

## comm\_no\_delay

**Parameter description:** Specifies whether to use the **NO\_DELAY** attribute of a communications library connection.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

---

**NOTICE**

If packet loss occurs in the cluster because a large number of packets are received per second, set this parameter to **off** so that small packets are combined into large packets for transmission to reduce the total number of packets.

---

**Default value:** off

## comm\_debug\_mode

**Parameter description:** Specifies whether to enable the debug mode of the TCP proxy communications library, that is, whether to print logs about the communication layer.

---

**NOTICE**

If this parameter is set to **on**, a huge number of logs will be printed, adding extra overhead and reducing database performance. Therefore, set it to **on** only in debugging scenarios.

---

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the debug logs of the communications library are printed.

- **off** indicates that the debug logs of the communications library are not printed.

**Default value:** off

## comm\_ackchk\_time

**Parameter description:** Specifies the duration after which the communications library server automatically triggers ACK when no data packet is received.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 20000. The unit is ms. **0:** disabled

**Default value:** 2000 (2s)

## comm\_timer\_mode

**Parameter description:** Specifies whether to enable the timer mode of the TCP proxy communications library, that is, whether to print timer logs in each phase of the communication layer.

---

### NOTICE

If this parameter is set to **on**, a huge number of logs will be printed, adding extra overhead and reducing database performance. Therefore, set it to **on** only in debugging scenarios.

---

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the timer logs of the communications library are printed.
- **off** indicates that the timer logs of the communications library are not printed.

**Default value:** off

## comm\_stat\_mode

**Parameter description:** Specifies whether to enable the statistics mode of the TCP proxy communications library, that is, whether to print statistics about the communication layer.

---

### NOTICE

If this parameter is set to **on**, a huge number of logs will be printed, adding extra overhead and reducing database performance. Therefore, set it to **on** only in debugging scenarios.

---

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the statistics logs of the communications library are printed.
- **off** indicates that the statistics logs of the communications library are not printed.

**Default value:** off

## enable\_stateless\_pooler\_reuse

**Parameter description:** Specifies whether to enable the reuse of the pooler connection pool. After the parameter is enabled, existing idle TCP connections can be reused. The setting takes effect after the cluster is restarted.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** or **true** indicates that the pooler reuse mode is enabled.
- **off** or **false** indicates that the pooler reuse mode is disabled.

---

### NOTICE

This parameter should be set to a same value on CNs and DN. If this parameter is set to **off** for CNs and **on** for DN, the cluster communication fails. Set this parameter to the same value for CNs and DN. Restart the cluster for the setting to take effect.

---

**Default value:** on

## comm\_cn\_dn\_logic\_conn

**Parameter description:** Specifies whether logical connections are used between CNs and DN. The setting takes effect after the cluster is restarted.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** or **true** indicates that the connections between CNs and DN are logical, with the libcomm component in use.
- **off** or **false** indicates that the connections between CNs and DN are physical, with the libpq component in use.

---

### NOTICE

Logical connections between CNs and DN are no longer supported. This parameter is provided for compatibility, but its value is fixed at **off**.

---

**Default value:** off

## COMM\_IPC

**Parameter description:** Specifies whether to print the packet sending and receiving status of each communication node.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** or **true** indicates that the function of logging packet sending and receiving status data is enabled.
- **off** or **false** indicates that the function of logging packet sending and receiving status data is disabled.

---

### NOTICE

```
set logging_module='on(COMM_IPC)'; --Enabled
set logging_module='off(COMM_IPC)'; --Disabled
show logging_module; -- View the setting result.
```

If this parameter is set to **on**, a huge number of logs will be printed, adding extra overhead and reducing database performance. Therefore, set it to **on** only in debugging scenarios and set it to **off** after debugging.

---

**Default value:** off

---

### NOTICE

```
set logging_module='on(COMM_IPC)'; --Enabled
set logging_module='off(COMM_IPC)'; --Disabled
show logging_module; -- View the setting result.
```

If this parameter is set to **on**, a huge number of logs will be printed, adding extra overhead and reducing database performance. Therefore, set it to **on** only in debugging scenarios and set it to **off** after debugging.

---

**Default value:** off

## COMM\_PARAM

**Parameter description:** Specifies whether to print the **session** parameter settings during node communication.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** or **true** indicates that the function of logging the **session** parameter settings is enabled.

- **off** or **false** indicates that the function of logging the **session** parameter settings is disabled.

---

**NOTICE**

```
set logging_module='on(COMM_PARAM)'; --Enabled
set logging_module='off(COMM_PARAM)'; --Disabled
show logging_module; -- View the setting result.
```

If this parameter is set to **on**, a huge number of logs will be printed, adding extra overhead and reducing database performance. Therefore, set it to **on** only in debugging scenarios and set it to **off** after debugging.

---

**Default value:** off

## 14.3.4 Resource Consumption

### 14.3.4.1 Memory

This section describes memory parameters.

---

**NOTICE**

These parameters, except **local\_syscache\_threshold**, take effect only after the database restarts.

---

#### memorypool\_enable

**Parameter description:** Specifies whether to enable a memory pool.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the memory pool is enabled.
- **off** indicates that the memory pool is disabled.

**Default value:** off

#### memorypool\_size

**Parameter description:** Specifies the memory pool size.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 128 x 1024 to 1073741823. The unit is KB.

**Default value:** 512 MB

## enable\_memory\_limit

**Parameter description:** Specifies whether to enable the logical memory management module.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the logical memory management module is enabled.
- **off** indicates that the logical memory management module is disabled.

**Default value:** on

---

### CAUTION

- Fixed overhead exists, that is, **shared\_buffers** and metadata (about 200 MB). If **max\_process\_memory** minus the fixed overhead is less than 2 GB, GaussDB forcibly sets **enable\_memory\_limit** to **off**. Metadata is the memory used in GaussDB and is related to some concurrent parameters, such as **max\_connections**, **thread\_pool\_attr**, and **max\_prepared\_transactions**.
  - If this parameter is set to **off**, the memory used by the database is not limited. When a large number of concurrent or complex queries are performed, too much memory is used, which may cause OS OOM problems.
- 

## max\_process\_memory

**Parameter description:** Specifies the maximum physical memory of a database node.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 2097152 to 2147483647. The unit is KB.

**Default value:**

Independent deployment: 360 GB (60-core CPU/480 GB memory); 192 GB (32-core CPU/256 GB memory); 96 GB (16-core CPU/128 GB memory); 40 GB (8-core CPU/64 GB memory); 20 GB (4-core CPU/32 GB memory); 10 GB (4-core CPU/16 GB memory)

**Setting suggestions:**

On DNs, the value of this parameter is determined based on the physical system memory and the number of primary DNs deployed on a single node. Parameter value = (Physical memory - **vm.min\_free\_kbytes**) x 0.7/(n + Number of primary DNs). This parameter is used to prevent node OOM caused by memory usage increase, ensuring system reliability. **vm.min\_free\_kbytes** indicates the OS memory reserved for the kernel to receive and send data. Its value is at least 5% of the total memory. Therefore, the value of **max\_process\_memory** is: Physical memory x 0.665/(n + Number of primary DNs). When the number of nodes in the cluster is less than or equal to 256, n = 1. When the number of nodes in the

cluster is greater than 256 and less than or equal to 512,  $n = 2$ . When the number of nodes in the cluster is greater than 512,  $n = 3$ .

You can set this parameter on CNs to the same value as that on DN.

RAM is the maximum memory allocated to the cluster during cluster planning. It equals the physical memory of servers.

---

 **CAUTION**

If this parameter is set to a value greater than the physical memory of the server, the OS OOM problem may occur.

---

## local\_syscache\_threshold

**Parameter description:** Specifies the size of system catalog cache in a session.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

- If **enable\_global\_plancache** is enabled, **local\_syscache\_threshold** must be set to a value greater than or equal to 16 MB to ensure that GPC takes effect.
- If **enable\_global\_syscache** and **enable\_thread\_pool** are enabled, this parameter indicates the total cache size of the current thread and sessions bound to the current thread.

**Value range:** an integer ranging from  $1 \times 1024$  to  $512 \times 1024$ . The unit is KB.

**Default value:**

- Independent deployment: 16 MB

## enable\_memory\_context\_control

**Parameter description:** Enables the function of checking whether the number of memory contexts exceeds the specified limit. This parameter applies only to the DEBUG version.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the function of checking the number of memory contexts is enabled.
- **off** indicates that the function of checking the number of memory contexts is disabled.

**Default value:** off

## uncontrolled\_memory\_context

**Parameter description:** Specifies which memory context will not be checked when the **enable\_memory\_context\_control** parameter is enabled. This parameter applies only to the DEBUG version.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

During the query, the title meaning string "MemoryContext white list:" is added to the beginning of the parameter value.

**Value range:** a string.

**Default value:** empty.

## shared\_buffers

**Parameter description:** Specifies the size of shared memory used by GaussDB. Increasing the value of this parameter causes GaussDB to request more System V shared memory than the default configuration allows.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 16 to 1073741823. The unit is 8 KB.

The minimum value changes according to **BLCKSZ**.

**Default value:**

Independent deployment:

CN: 4 GB (60-core CPU/480 GB memory); 2 GB (32-core CPU/256 GB memory, 16-core CPU/128 GB memory); 1 GB (8-core CPU/64 GB memory); 512 MB (4-core CPU/32 GB memory) 256 MB (4-core CPU/16 GB memory)

DN: 140 GB (60-core CPU/480 GB memory); 76 GB (32-core CPU/256 GB memory); 40 GB (16-core CPU/128 GB memory); 16 GB (8-core CPU/64 GB memory); 8 GB (4-core CPU/32 GB memory); 4 GB (4-core CPU/16 GB memory)

**Setting suggestions:**

1. Set this parameter on DNs to a value greater than that on CNs because most queries in GaussDB are pushed down.
2. Set **shared\_buffers** to a value less than 40% of the memory.
3. If **shared\_buffers** is set to a larger value, increase the value of **checkpoint\_segments** because a longer period of time is required to write a large amount of new or changed data.
4. If the process fails to be restarted after the value of **shared\_buffers** is changed, perform either of the following operations based on the error information:
  - a. Adjust the **kernel.shmall**, **kernel.shmmax**, and **kernel.shmmin** OS parameters. For details, see "Preparing for Installation > Modifying OS Configuration > Configuring Other OS Parameters" in *Installation Guide*.
  - b. Run the **free -g** command to check whether the available memory and swap space of the OS are sufficient. If the memory is insufficient, manually stop other user programs that occupy much memory.
  - c. Do not set **shared\_buffers** to an excessively large or small value.

## segment\_buffers

**Parameter description:** This parameter is reserved and is not supported currently.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

## bulk\_write\_ring\_size

**Parameter description:** Specifies the size of a ring buffer used for parallel data import.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 16384 to 2147483647. The unit is KB.

**Default value:** 2 GB

**Setting suggestions:** Increase the value of this parameter on DNs if a huge amount of data will be imported.

## standby\_shared\_buffers\_fraction

**Parameter description:** Specifies the **shared\_buffers** proportion used on the server where a standby instance is deployed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a double-precision floating-point number ranging from 0.1 to 1.0

**Default value:** 1

## temp\_buffers

**Parameter description:** Specifies the maximum size of local temporary buffers used by a database session.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

This parameter can be modified only before the first use of temporary tables within each session. Subsequent attempts to change the value of this parameter will not take effect on that session.

A session allocates temporary buffers based on the value of **temp\_buffers**. If a large value is set in a session that does not require many temporary buffers, only the overhead of one buffer descriptor is added. If a buffer is used, additional 8192 bytes will be consumed for it.

**Value range:** an integer ranging from 100 to 1073741823. The unit is 8 KB.

**Default value:** 1 MB

## max\_prepared\_transactions

**Parameter description:** Specifies the maximum number of transactions that can stay in the **prepared** state simultaneously. Increasing the value of this parameter

causes GaussDB to request more System V shared memory than the default configuration allows.

When GaussDB is deployed as an HA system, set this parameter on standby servers to a value greater than or equal to that on primary servers. Otherwise, queries will fail on the standby servers.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 262143.

**Default value:**

- Independent deployment:  
1200 (60-core CPU/480 GB memory and 32-core CPU/256 GB memory); 800 (16-core CPU/128 GB memory); 400 (8-core CPU/64 GB memory); 300 (4-core CPU/32 GB memory); 200 (4-core CPU/16 GB memory)

 **NOTE**

To avoid failures in the preparation step, the value of this parameter must be greater than the number of worker threads in `thread_pool_attr` in thread pool mode. In non-thread pool mode, the value of this parameter must be greater than or equal to the value of `max_connections`.

## work\_mem

**Parameter description:** Specifies the amount of memory to be used by internal sort operations and hash tables before they write data into temporary disk files. Sort operations are required for **ORDER BY**, **DISTINCT**, and merge joins. Hash tables are used in hash joins, hash-based aggregation, and hash-based processing of **IN** subqueries.

In a complex query, several sort or hash operations may run in parallel; each operation will be allowed to use as much memory as this parameter specifies. If the memory is insufficient, data will be written into temporary files. In addition, several running sessions could be performing such operations concurrently. Therefore, the total memory used may be many times the value of **work\_mem**.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 64 to 2147483647. The unit is KB.

**Default value:**

- Independent deployment:  
128 MB (60-core CPU/480 GB memory, 32-core CPU/256 GB memory, and 16-core CPU/128 GB memory); 64 MB (8-core CPU/64 GB memory); 32 MB (4-core CPU/32 GB memory); 16 MB (4-core CPU/16 GB memory)

---

**NOTICE**

**Setting suggestions:**

If the physical memory specified by **work\_mem** is insufficient, additional operator calculation data will be written into temporary tables based on query characteristics and the degree of parallelism. This reduces performance by five to ten times, and prolongs the query response time from seconds to minutes.

- For complex serial queries, each query requires five to ten associated operations. Set **work\_mem** using the following formula: **work\_mem** = 50% of the memory/10.
  - For simple serial queries, each query requires two to five associated operations. Set **work\_mem** using the following formula: **work\_mem** = 50% of the memory/5.
  - For concurrent queries, set **work\_mem** using the following formula: **work\_mem** = **work\_mem** for serial queries/Number of concurrent SQL statements.
  - BitmapScan hash tables are also restricted by **work\_mem**, but will not be forcibly flushed to disks. In the case of complete lossify, every 1-MB memory occupied by the hash table corresponds to a 16 GB page of BitmapHeapScan. After the upper limit of **work\_mem** is reached, the memory increases linearly with the data access traffic based on this ratio.
- 

## query\_mem

**Parameter description:** Specifies the memory used by a query.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or an integer greater than 32 MB. The default unit is KB.

**Default value:** 0

---

**NOTICE**

- If the value of **query\_mem** is greater than 0, the optimizer adjusts the memory cost estimate to this value when generating an execution plan.
  - If the value is set to a negative value or a positive integer less than 32 MB, the default value 0 is used. In this case, the optimizer does not adjust the estimated query memory.
- 

## query\_max\_mem

**Parameter description:** Specifies the maximum memory that can be used by a query.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or an integer greater than 32 MB. The default unit is KB.

**Default value:** 0

---

**NOTICE**

- If the value of **query\_max\_mem** is greater than 0, an error is reported when the query memory usage exceeds the value.
  - If the value is set to a negative value or a positive integer less than 32 MB, the default value 0 is used. In this case, the optimizer does not limit the query memory.
- 

## **maintenance\_work\_mem**

**Parameter description:** Specifies the maximum amount of memory to be used by maintenance operations, such as **VACUUM** and **CREATE INDEX**. This parameter may affect the execution efficiency of **VACUUM**, **VACUUM FULL**, **CLUSTER**, and **CREATE INDEX**.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1024 to 2147483647. The unit is KB.

**Default value:**

- Independent deployment:  
CN: 1 GB (60-core CPU/480 GB memory); 512 MB (32-core CPU/256 GB memory); 256 MB (16-core CPU/128 GB memory); 128 MB (8-core CPU/64 GB memory); 64 MB (4-core CPU/32 GB memory); 32 MB (4-core CPU/16 GB memory)  
DN: 2 GB (60-core CPU/480 GB memory); 1 GB (32-core CPU/256 GB memory); 512 MB (16-core CPU/128 GB memory); 256 MB (8-core CPU/64 GB memory); 128 MB (4-core CPU/32 GB memory); 64 MB (4-core CPU/16 GB memory)

---

**NOTICE**

**Setting suggestions:**

- The value of this parameter must be greater than that of **work\_mem** so that database dumps can be cleared or restored more quickly. In a database session, only one maintenance operation can be performed at a time. Maintenance is usually performed when there are not many running sessions.
  - When the **Automatic Vacuuming** process is running, up to **autovacuum\_max\_workers** times this memory may be allocated. In this case, set **maintenance\_work\_mem** to a value greater than or equal to that of **work\_mem**.
  - If a large amount of data is to be clustered, increase the value of this parameter in the session.
-

## max\_stack\_depth

**Parameter description:** Specifies the maximum safe depth of the GaussDB execution stack. The safety margin is required because the stack depth is not checked in every routine in the server, but only in key potentially-recursive routines, such as expression evaluation.

This parameter is a SUSE parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 100 to 2147483647. The unit is KB.

**Default value:**

- If the value of **ulimit -s** minus 640 KB is greater than or equal to 2 MB, the default value of this parameter is 2 MB.
- If the value of **ulimit -s** minus 640 KB is less than 2 MB, the default value of this parameter is the value of **ulimit -s** minus 640 KB.

---

### NOTICE

When setting this parameter, comply with the following principles:

- The database needs to reserve 640 KB stack depth. Therefore, the maximum value of this parameter is the actual stack size limit enforced by the OS kernel (as set by **ulimit -s**) minus 640 KB.
  - If the value of this parameter is greater than the value of **ulimit -s** minus 640 KB before the database is started, the database fails to be started. During database running, if the value of this parameter is greater than the value of **ulimit -s** minus 640 KB, this parameter does not take effect.
  - If the value of **ulimit -s** minus 640 KB is less than the minimum value of this parameter, the database fails to be started.
  - Setting this parameter to a value greater than the actual kernel limit means that a running recursive function may crash an individual backend process.
  - Since not all OSs provide this function, you are advised to set a specific value for this parameter.
  - The default value is 2 MB, which is relatively small and does not easily cause system breakdown.
- 

## bulk\_read\_ring\_size

**Parameter description:** Specifies the ring buffer size used for parallel data export.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 256 to 2147483647. The unit is KB.

**Default value:** 16 MB

## enable\_early\_free

**Parameter description:** Specifies whether the operator memory can be released in advance.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the operator memory can be released in advance.
- **off** indicates that the operator memory cannot be released in advance.

**Default value:** on

## memory\_trace\_level

**Parameter description:** Specifies the control level for recording memory allocation information after the dynamic memory usage exceeds 90% of the maximum dynamic memory. This parameter takes effect only when the GUC parameters **use\_workload\_manager** and **enable\_memory\_limit** are enabled. This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **none:** indicates that memory application information is not recorded.
- **level1:** After the dynamic memory usage exceeds 90% of the maximum dynamic memory, the following information is recorded and the recorded memory information is saved in the *\$GAUSSLOG/mem\_log* directory.
  - Global memory overview.
  - Memory usage of the top 20 memory contexts of the instance, session, and thread types.
  - The **totalsize** and **freesize** columns for each memory context.
- **level2:** After the dynamic memory usage exceeds 90% of the maximum dynamic memory, the following information is recorded and the recorded memory information is saved in the *\$GAUSSLOG/mem\_log* directory.
  - Global memory overview.
  - Memory usage of the top 20 memory contexts of the instance, session, and thread types.
  - The **totalsize** and **freesize** columns for each memory context.
  - Detailed information about all memory applications in each memory context, including the file where the allocated memory is located, line number, and size.

**Default value:** level1

#### NOTICE

- If this parameter is set to **level2**, the memory allocation details (file, line, and size) of each memory context are recorded, which greatly affects the performance. Therefore, exercise caution when setting this parameter.
- You can use the system function **gs\_get\_history\_memory\_detail(cstring)** to query the recorded memory snapshot information. For details about the function, see "SQL Reference > Functions and Operators > Statistics Functions" in *Developer Guide*.
- If the **use\_workload\_manager** parameter is disabled and the **bypass\_workload\_manager** parameter is enabled, this parameter also takes effect. The **bypass\_workload\_manager** parameter is of the SIGHUP type; therefore, after the reload mode is set, you need to restart the database for the setting to take effect.
- The recorded memory context is obtained after all memory contexts of the same type with the same name are summarized.

## resilience\_memory\_reject\_percent

**Parameter description:** Specifies the dynamic memory usage percentage for escape from memory overload. This parameter takes effect only when the GUC parameters **use\_workload\_manager** and **enable\_memory\_limit** are enabled. This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, consisting of one or more characters.

This parameter consists of **recover\_memory\_percent** and **overload\_memory\_percent**. The meanings of the two parts are as follows:

- **recover\_memory\_percent:** Percentage of the dynamic memory usage when the memory recovers from overload to the maximum dynamic memory. When the dynamic memory usage is less than the maximum dynamic memory multiplied by the value of this parameter, the overload escape function is disabled and new connections are allowed. The value ranges from 0 to 100. The value indicates a percentage.
- **overload\_memory\_percent:** Percentage of the dynamic memory usage to the maximum dynamic memory when the memory is overloaded. When the dynamic memory usage is greater than the maximum dynamic memory multiplied by the value of this parameter, the current memory is overloaded. In this case, the overload escape function is triggered to kill sessions and new connections are prohibited. The value ranges from 0 to 100. The value indicates a percentage.

**Default value:** '0,0', indicating that the escape from memory overload function is disabled.

#### Example:

```
resilience_memory_reject_percent = '70,90'
```

When the memory usage exceeds 90% of the upper limit, new connections are forbidden and stacked sessions are killed. When the memory usage is less than

70% of the upper limit, session killing is stopped and new connections are allowed.

---

#### NOTICE

- You can query the maximum dynamic memory and used dynamic memory in the **pv\_total\_memory\_detail** view. **max\_dynamic\_memory** indicates the maximum dynamic memory, and **dynamic\_used\_memory** indicates the used dynamic memory.
  - If this parameter is set to a small value, the escape from memory overload process is frequently triggered. As a result, ongoing sessions are forcibly logged out, and new connections fail to be connected for a short period of time. Therefore, exercise caution when setting this parameter based on the actual memory usage.
  - If the **use\_workload\_manager** parameter is disabled and the **bypass\_workload\_manager** parameter is enabled, this parameter also takes effect. The **bypass\_workload\_manager** parameter is of the SIGHUP type; therefore, after the reload mode is set, you need to restart the database for the setting to take effect.
  - The values of **recover\_memory\_percent** and **overload\_memory\_percent** can be **0** at the same time. In addition, the value of **recover\_memory\_percent** must be smaller than that of **overload\_memory\_percent**. Otherwise, the setting does not take effect.
- 

## resilience\_escape\_user\_permissions

**Parameter description:** Specifies the escape permission of users. You can set it for multiple users and separate users by commas (.). The value **sysadmin** indicates that jobs of user **sysadmin** can be canceled by the escape function. The value **monadmin** indicates that jobs of user **monadmin** can be canceled by the escape function. By default, this parameter is left blank, indicating that the escape function of the sysadmin and monadmin users is disabled. The value can only be **sysadmin**, **monadmin**, or an empty string. This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, consisting of one or more characters.

Currently, this parameter supports only three values: **sysadmin**, **monadmin**, and "". The meanings of these values are as follows:

- **sysadmin:** Jobs of user **sysadmin** can be canceled by the escape function.
- **monadmin:** Jobs of user **monadmin** can be canceled by the escape function.
- **":** The escape function of the sysadmin and monadmin users is disabled.

**Default value:** "", indicating that the escape function of the sysadmin and monadmin users is disabled.

#### Example:

```
resilience_escape_user_permissions = 'sysadmin,monadmin'
```

The escape function is enabled for both the sysadmin and monadmin users.

---

**NOTICE**

- You can set this parameter to multiple values separated by commas (,), for example, **resilience\_escape\_user\_permissions = 'sysadmin,monadmin'**. You can also set this parameter to only one value, for example, **resilience\_escape\_user\_permissions = 'monadmin'**.
  - If this parameter is set for multiple times, the latest setting takes effect.
  - If this parameter is set to any value in the value range, common users support the escape function.
  - If a user has both the sysadmin and monadmin role permissions, the escape function of the user can be triggered only when **resilience\_escape\_user\_permissions** is set to **'sysadmin,monadmin'**.
- 

### 14.3.4.2 Disk Space

This section describes the disk space parameters, which are used to set limits on the disk space for storing temporary files.

#### sql\_use\_spacelimit

**Parameter description:** Specifies the space size for files to be flushed to disks when a single SQL statement is executed on a single DN. The managed space includes the space occupied by ordinary tables, temporary tables, and intermediate result sets to be flushed to disks. This parameter does not take effect for initial users.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 2147483647. The unit is KB. -1 indicates no limit.

**Default value:** -1

#### temp\_file\_limit

**Parameter description:** Specifies the limit on the size of a temporary file spilled to disk in a session. The temporary file can be a sort or hash temporary file, or the storage file for a held cursor.

This is a session-level setting.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

This parameter does not apply to disk space used for temporary tables during the SQL query process.

---

**Value range:** an integer ranging from -1 to 2147483647. The unit is KB. -1 indicates no limit.

**Default value:** -1

### 14.3.4.3 Kernel Resource Usage

This section describes kernel resource parameters. Whether these parameters take effect depends on OS settings.

#### **max\_files\_per\_process**

**Parameter description:** Specifies the maximum number of simultaneously open files allowed by each server process. If the kernel is enforcing a proper limit, setting this parameter is not required.

But on some platforms, especially on most BSD systems, the kernel allows independent processes to open far more files than the system can really support. If the message "Too many open files" is displayed, set this parameter to a smaller value. Generally, the system must meet this requirement: Number of file descriptors  $\geq$  Maximum number of concurrent statements  $\times$  Number of primary DNS of the current server  $\times$  **max\_files\_per\_process**  $\times$  3

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 25 to 2147483647

**Default value:** 1024

#### **shared\_preload\_libraries**

**Parameter description:** Specifies one or more shared libraries to be preloaded at server start. If multiple libraries are to be loaded, separate their names using commas (,). Only the **sysadmin** user can access this parameter. For example, **\$libdir/mylib** will cause **mylib.so** (or on some platforms, **mylib.sl**) to be preloaded before the loading of the standard library directory.

You can preinstall the GaussDB stored procedure library using the **\$libdir/plXXX** syntax as described in the preceding text. **XXX** can only be **pgsql**, **perl**, **tcl**, or **python**.

By preloading a shared library and initializing it as required, the library startup time is avoided when the library is first used. However, the time to start each new server process may increase, even if that process never uses the library. Therefore, set this parameter only for libraries that will be used in most sessions.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

#### NOTICE

- If a specified library is not found, the GaussDB service will fail to start.
  - Each GaussDB-supported library has a special mark that is checked to guarantee compatibility. Therefore, libraries that do not support GaussDB cannot be loaded in this way.
-

**Value range:** a string

**Default value:** `security_plugin`

#### 14.3.4.4 Cost-based Vacuum Delay

This feature allows administrators to reduce the I/O impact of the **VACUUM** and **ANALYZE** statements on concurrent database activities. It is often more important to prevent maintenance statements, such as **VACUUM** and **ANALYZE**, from affecting other database operations than to run them quickly. Cost-based vacuum delay provides a way for administrators to achieve this purpose.

---

#### NOTICE

Certain vacuum operations hold critical locks and should be complete as quickly as possible. In GaussDB, cost-based vacuum delays do not take effect during such operations. To avoid uselessly long delays in such cases, the actual delay is the larger of the two calculated values:

- $\text{vacuum\_cost\_delay} \times \text{accumulated\_balance} / \text{vacuum\_cost\_limit}$
  - $\text{vacuum\_cost\_delay} \times 4$
- 

### Background

During the execution of the **ANALYZE | ANALYSE** and **VACUUM** statements, the system maintains an internal counter that keeps track of the estimated cost of the various I/O operations that are performed. For details about **ANALYZE | ANALYSE** and **VACUUM**, see the corresponding sections in "SQL Reference > SQL Syntax" in *Developer Guide*. When the accumulated cost reaches a limit (specified by **vacuum\_cost\_limit**), the process performing the operation will sleep for a short period of time (specified by **vacuum\_cost\_delay**). Then, the counter resets and the operation continues.

By default, this feature is disabled. To enable this feature, set **vacuum\_cost\_delay** to a positive value.

### **vacuum\_cost\_delay**

**Parameter description:** Specifies the length of time that a process will sleep when **vacuum\_cost\_limit** has been exceeded.

On many systems, the effective resolution of the sleep length is 10 milliseconds. Therefore, setting this parameter to a value that is not a multiple of 10 has the same effect as setting it to the next higher multiple of 10.

This parameter is usually set to a small value, such as 10 or 20 milliseconds. Adjusting vacuum's resource consumption is best done by changing other vacuum cost parameters.

This parameter is a **USERSET** parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 100. The unit is ms. A positive number enables cost-based vacuum delay and **0** disables cost-based vacuum delay.

**Default value:** 0

### **vacuum\_cost\_page\_hit**

**Parameter description:** Specifies the estimated cost for vacuuming a buffer found in the shared buffer. It represents the cost to lock the buffer pool, look up the shared hash table, and scan the content of the page.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 10000

**Default value:** 1

### **vacuum\_cost\_page\_miss**

**Parameter description:** Specifies the estimated cost for vacuuming a buffer read from the disk. It represents the cost to lock the buffer pool, look up the shared hash table, read the desired block from the disk, and scan the block.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 10000

**Default value:** 10

### **vacuum\_cost\_page\_dirty**

**Parameter description:** Specifies the estimated cost charged when vacuum modifies a block that was previously clean. It represents the extra cost required to update the dirty block out to the disk again.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 10000

**Default value:** 20

### **vacuum\_cost\_limit**

**Parameter description:** Specifies the cost limit. The vacuuming process will sleep if this limit is exceeded.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 10000

**Default value:** 1000

## **14.3.4.5 Background Writer**

This section describes background writer parameters. The background writer process is used to write dirty data (new or modified data) in shared buffers to

disks. This mechanism ensures that database processes seldom or never need to wait for a write action to occur when handling user queries.

It also mitigates performance deterioration caused by checkpoints because only a few of dirty pages need to be flushed to the disk when the checkpoints arrive. This mechanism, however, increases the overall net I/O load because while a repeatedly-dirtied page may otherwise be written only once per checkpoint interval, the background writer may write it several times as it is dirtied in the same interval. In most cases, continuous light loads are preferred, instead of periodic load peaks. The parameters discussed in this section can be set based on actual requirements.

## bgwriter\_delay

**Parameter description:** Specifies the interval at which the background writer writes dirty shared buffers. Each time, the backend write process initiates write operations for some dirty buffers. In full checkpoint mode, the **bgwriter\_lru\_maxpages** parameter is used to control the amount of data to be written each time, and the process is restarted after *bgwriter\_delay* ms hibernation. In incremental checkpoint mode, the number of target idle buffer pages is calculated based on the value of **candidate\_buf\_percent\_target**. If the number of idle buffer pages is insufficient, a batch of pages is flushed to disks every *bgwriter\_delay* ms. The number of flushed pages is calculated based on the target difference percentage. The maximum number of flushed pages is limited by **max\_io\_capacity**.

In many systems, the effective resolution of sleep delays is 10 milliseconds. Therefore, setting this parameter to a value that is not a multiple of 10 has the same effect as setting it to the next higher multiple of 10.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10 to 10000. The unit is millisecond.

**Default value:** 2s

**Setting suggestion:** Reduce this value in slow data writing scenarios to reduce the checkpoint load.

## candidate\_buf\_percent\_target

**Parameter description:** Specifies the expected percentage of available buffers in the shared\_buffer memory buffer in the candidate buffer chain when the incremental checkpoint is enabled. If the number of available buffers in the current candidate chain is less than the target value, the bgwriter thread starts flushing dirty pages that meet the requirements.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a double-precision floating point number ranging from 0.1 to 0.85

**Default value:** 0.3

## bgwriter\_lru\_maxpages

**Parameter description:** Specifies the number of dirty buffers the background writer can write in each round.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1000

### NOTE

When this parameter is set to **0**, the background writer is disabled. This setting does not affect checkpoints.

**Default value:** 100

## bgwriter\_lru\_multiplier

**Parameter description:** Specifies the coefficient used to estimate the number of dirty buffers the background writer can write in the next round.

The number of dirty buffers written in each round depends on the number of buffers used by server processes during recent rounds. The estimated number of buffers required in the next round is calculated using the following formula: Average number of recently used buffers x **bgwriter\_lru\_multiplier**. The background writer writes dirty buffers until sufficient, clean and reusable buffers are available. The number of buffers the background writer writes in each round is always less than or equal to the value of **bgwriter\_lru\_maxpages**.

Therefore, the value **1.0** of **bgwriter\_lru\_multiplier** represents a just-in-time policy of writing exactly the number of dirty buffers predicted to be required. Larger values provide some cushion against spikes in demand, whereas smaller values intentionally leave more writes to be done by server processes.

Smaller values of **bgwriter\_lru\_maxpages** and **bgwriter\_lru\_multiplier** reduce the extra I/O load caused by the background writer, but make it more likely that server processes will have to issue writes for themselves, delaying interactive queries.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0 to 10

**Default value:** 2

## pagewriter\_thread\_num

**Parameter description:** Specifies the number of threads for background page flushing after the incremental checkpoint is enabled. Dirty pages are flushed in sequence to disks, promoting recovery points.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 16

**Default value:** 4

## dirty\_page\_percent\_max

**Parameter description:** Specifies the percentage of dirty pages to **shared\_buffers** after the incremental checkpoint is enabled. When the value of this parameter is reached, the background page flush thread flushes dirty pages based on the maximum value of **max\_io\_capacity**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0.1 to 1

**Default value:** 0.9

## pagewriter\_sleep

**Parameter description:** Specifies the interval for the pagewriter thread to flush dirty pages to disks after the incremental checkpoint is enabled. When the ratio of dirty pages to **shared\_buffers** reaches **dirty\_page\_percent\_max**, the number of pages in each batch is calculated based on the value of **max\_io\_capacity**. In other cases, the number of pages in each batch decreases proportionally.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 3600000. The unit is ms.

**Default value:** 2000 ms (2s)

## max\_io\_capacity

**Parameter description:** Specifies the I/O upper limit per second for the backend write process to flush pages in batches. Set this parameter based on the service scenario and the disk I/O capability. If the RTO is short or the data volume is much larger than the shared memory and the service access data volume is random, the value of this parameter cannot be too small. A small value of **max\_io\_capacity** reduces the number of pages flushed by the backend write process. If a large number of pages are evicted due to service triggering, the services are affected.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 30720 to 10485760. The unit is KB.

**Default value:** 500 MB/s

## enable\_consider\_usecount

**Parameter description:** Specifies whether the backend thread considers the page popularity during page replacement. You are advised to enable this parameter in large-capacity scenarios.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on/true:** The page popularity is considered.
- **off/false:** The page popularity is not considered.

**Default value:** off

## dw\_file\_num

**Parameter description:** Specifies the number of doublewrite files to be written in batches. The value is related to **pagewriter\_thread\_num** and cannot be greater than **pagewriter\_thread\_num**. If the value is too large, it will be corrected to the value of **pagewriter\_thread\_num**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 16

**Default value:** 1

## dw\_file\_size

**Parameter description:** Specifies the size of each doublewrite file.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer, in the range [32,256]

**Default value:** 256

### 14.3.4.6 Asynchronous I/O

## checkpoint\_flush\_after

**Parameter description:** Specifies the threshold for the number of pages flushed by the checkpointer thread. If the threshold is exceeded, the operating system is instructed to flush the pages cached in the operating system asynchronously. In GaussDB, the disk page size is 8 KB.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. The size of a single page is 8 KB. For example, if the value is **32**, the checkpointer thread continuously writes 32 disk pages (that is,  $32 \times 8 = 256$  KB) before asynchronous flush.

**Default value:** 256 KB (32 pages)

## bgwriter\_flush\_after

**Parameter description:** Specifies the threshold for the number of pages flushed by the background writer thread. If the number of pages exceeds the threshold, the background writer thread instructs the operating system to asynchronously flush the pages cached in the operating system to disks. In GaussDB, the disk page size is 8 KB.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. The size of a single page is 8 KB. For example, if the value is **64**, the background writer thread continuously writes 64 disk pages (that is,  $64 \times 8 = 512$  KB) before asynchronous flush.

**Default value:** 512 KB (64 pages)

## backend\_flush\_after

**Parameter description:** Specifies the threshold for the number of pages flushed by the backend thread. If the number of pages exceeds the threshold, the backend thread instructs the operating system to asynchronously flush the pages cached in the operating system to disks. In GaussDB, the disk page size is 8 KB.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. For example, if the value is **64**, the backend thread continuously writes 64 disk pages (that is,  $64 \times 8 = 512$  KB) before asynchronous flush.

**Default value:** 0

## 14.3.5 Parallel Data Import

GaussDB provides a parallel data import function that enables a large amount of data to be imported in a fast and efficient manner. This section describes parameters for importing data to GaussDB in parallel.

### raise\_errors\_if\_no\_files

**Parameter description:** Specifies whether to distinguish between the problems "the number of imported file records is empty" and "the imported file does not exist". If this parameter is set to **TRUE** and the problem "the imported file does not exist" occurs, GaussDB will report the error message "file does not exist".

This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the messages of "the number of imported file records is empty" and "the imported file does not exist" are distinguished when files are imported.
- **off** indicates that the messages of "the number of imported file records is empty" and "the imported file does not exist" are the same when files are imported.

**Default value:** off

### gds\_debug\_mod

**Parameter description:** Specifies whether to enable the debug function of Gauss Data Service (GDS). This parameter is used to better locate and analyze GDS

faults. After the debug function is enabled, types of packets received or sent by GDS, peer end of GDS during command interaction, and other interaction information about GDS are written into the logs of corresponding nodes in the cluster. In this way, the state switching on the GaussDB state machine and the current state are recorded. If this function is enabled, additional log I/O resources will be consumed, affecting log performance and validity. You are advised to enable this function only when locating GDS faults.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:**

- **on** indicates that the GDS debug function is enabled.
- **off** indicates that the GDS debug function is disabled.

**Default value:** off

## safe\_data\_path

**Parameter description:** Specifies the path prefix restriction except for the initial user. Currently, the path prefix restriction applies to the COPY operation and advanced packages.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string of up to 4,096 characters

**Default value:** NULL

---

### CAUTION

- If a soft link file exists in the **safe\_data\_path** directory, the system processes the file based on the actual file path to which the soft link points. If the actual path is not in the **safe\_data\_path** directory, an error is reported.
  - If a hard link file exists in the **safe\_data\_path** directory, it can be used properly. For security purposes, exercise caution when using hard link files. Do not create hard link files that point to other directories in the **safe\_data\_path** directory. Ensure that the permission on the **safe\_data\_path** directory is minimized.
- 

## enable\_copy\_server\_files

**Parameter description:** Specifies whether to enable the permission to copy server files.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the permission to copy server files is enabled.
- **off** indicates that the permission to copy server files is disabled.

**Default value:** off

---

**NOTICE**

When the **enable\_copy\_server\_files** parameter is disabled, only the initial user is allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement. When the **enable\_copy\_server\_files** parameter is enabled, users with the **SYSADMIN** permission or users who inherit the **gs\_role\_copy\_files** permission of the built-in role are allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement.

---

## 14.3.6 Write Ahead Log

### 14.3.6.1 Settings

#### wal\_level

**Parameter description:** Specifies the level of information to be written to the WAL. The value cannot be empty or commented out.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- To enable WAL archiving and data streaming replication between primary and standby servers, set this parameter to **archive**, **hot\_standby**, or **logical**.
  - If this parameter is set to **archive** or **minimal**, **hot\_standby** must be set to **off**. In a distributed environment, **hot\_standby** cannot be set to **off**. Therefore, you are not advised to set this parameter to **archive** or **minimal**. Otherwise, the database cannot be started.
- 

**Value range:** enumerated values

- **minimal**  
Advantages: Certain bulk operations (including creating tables and indexes, executing cluster operations, and copying tables) are safely skipped in logging, which can make those operations much faster.  
Disadvantages: WALs contain only basic information required for recovery from a database server crash or an emergency shutdown. Data cannot be restored from archived WALs.
- **archive**  
Adds logging required for WAL archiving, supporting the database restoration from archives.
- **hot\_standby**
  - Further adds information required to run SQL queries on a standby server and takes effect after a server restart.
  - To enable read-only queries on a standby server, the **wal\_level** parameter must be set to **hot\_standby** on the primary server and the same value

must be set on the standby server. There are few measurable differences in performance between using **hot\_standby** and **archive** levels. However, feedback is welcome if any differences in their impacts on product performance are noticeable.

- **logical**

Only when this parameter is set to **logical**, logical logs can be parsed and the primary key information is recorded in Xlogs.

**Default value:** hot\_standby

## fsync

**Parameter description:** Specifies whether the GaussDB server uses the **fsync()** function (see [wal\\_sync\\_method](#)) to ensure that updates can be written to disks in a timely manner.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- Using the **fsync()** function ensures that the data can be recovered to a known state when an OS or a hardware crashes.
- Setting this parameter to **off** may result in unrecoverable data corruption in a system crash.

---

**Value range:** Boolean

- **on** indicates that the **fsync()** function is used.
- **off** indicates that the **fsync()** function is not used.

**Default value:** on

## synchronous\_commit

**Parameter description:** Specifies the synchronization mode of the current transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

Generally, logs generated by a transaction are synchronized in the following sequence:

1. The primary node writes the logs to the local memory.
2. The primary node writes the logs in the local memory to the local file system.
3. The primary node flushes the logs in the local file system to disks.
4. The primary node sends the logs to the standby node.
5. The standby node receives the logs and saves them to its local memory.
6. The standby node writes the logs in the local memory to the local file system.
7. The standby node flushes the logs in the local file system to disks.

8. The standby node replays the logs to complete the incremental update of data files.

**Value range:** enumerated values

- **on (true, yes, 1):** The primary node waits for the standby node to flush logs to disks before committing a transaction.
- **off (false, no, 0):** The primary node commits a transaction without waiting for the primary node to flush logs to disks. This mode is also called asynchronous commit.
- **local:** The primary node waits for the primary node to flush logs to disks before committing a transaction. This mode is also called local commit.
- **remote\_write:** The primary node waits for the standby node to write logs to the file system before committing a transaction. (The logs do not need to be flushed to disks.)
- **remote\_receive:** The primary node waits for the standby node to receive logs before committing a transaction. (The logs do not need to be written to the file system.)
- **remote\_apply:** The primary node waits for the standby node to complete log replay before committing a transaction.
- **true:** same as **on**.
- **false:** same as **off**.
- **yes:** same as **on**.
- **no:** same as **off**.
- **1:** same as **on**.
- **0:** same as **off**.
- **2:** same as **remote\_apply**.

**Default value:** on

## wal\_sync\_method

**Parameter description:** Specifies the method used for forcing WAL updates out to disk.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

If **fsync** is set to **off**, the setting of this parameter does not take effect because WAL file updates will not be forced out to disk.

---

**Value range:** enumerated values

- **open\_datasync** indicates that WAL files are opened with the **O\_DSYNC** option.
- **fdatsync** indicates that **fdatsync()** is called at each commit. (SLES 10 and SLES 11 are supported.)

- **fsync\_writethrough** indicates that **fsync()** is called at each commit to force data in the buffer to be written to the disk.

 NOTE

**wal\_sync\_method** can be set to **fsync\_writethrough** on a Windows platform, but this setting has the same effect as setting the parameter to **fsync** on the Windows platform.

- **fsync** indicates that **fsync()** is called at each commit. (SLES 10 and SLES 11 are supported.)
- **open\_sync** indicates that the **open()** with the O\_SYNC option is used to write WAL files (SLES 10 and SLES 11 are supported).

 NOTE

Not all platforms support the preceding parameters.

**Default value:** **fdatasync**

## full\_page\_writes

**Parameter description:** Specifies whether the GaussDB server writes the entire content of each disk page to WALs during the first modification of that page after a checkpoint.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- This parameter is needed because a page write that is in process during an OS crash might be only partially completed, leading to an on-disk page that contains a mix of old and new data. The row-level change data normally stored in WALs will not be enough to completely restore such a page during post-crash recovery. Storing the full page image guarantees that the page can be correctly restored, but at the price of increasing the amount of data that must be written to WALs.
- Setting this parameter to **off** might lead to unrecoverable data corruption after a system failure. It might be safe to set this parameter to **off** if you have hardware (such as a battery-backed disk controller) or file-system software (such as ReiserFS 4) that reduces the risk of partial page writes to an acceptably low level.

---

**Value range:** Boolean

- **on** indicates that this feature is enabled.
- **off** indicates that this feature is disabled.

**Default value:** **on**

## wal\_log\_hints

**Parameter description:** Specifies whether to write an entire page to WALs during the first modification of that page after a checkpoint, even for non-critical modifications of so-called hint bits. You are advised not to modify the setting.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the entire page is written to WALs.
- **off** indicates that the entire page is not written to WALs.

**Default value:** on

## wal\_buffers

**Parameter description:** Specifies the number of **XLOG\_BLCKSZ** used for storing WAL data. The size of each **XLOG\_BLCKSZ** is 8 KB.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:**  $-1$  to  $2^{18}$ . The minimum value is  $-1$  and the maximum value is 262144. The unit is 8 KB.

- If this parameter is set to  $-1$ , the value of **wal\_buffers** is automatically adjusted based on the value of **shared\_buffers**. The default value is 1/32 of **shared\_buffers**. If the value is less than 8, it will be forcibly set to **8**. If the value is greater than 2048, it will be forcibly set to **2048**.
- If this parameter is set to a value other than  $-1$  and smaller than **4**, the value **4** is forcibly used.
- Independent deployment: 1 GB (60-core CPU/480 GB memory and 32-core CPU/256 GB memory); 512 MB (16-core CPU/128 GB memory); 256 MB (8-core CPU/64 GB memory); 128 MB (4-core CPU/32 GB memory); 64 MB (4-core CPU/16 GB memory)

**Setting suggestions:** The content of the WAL buffer is written to disks at every transaction commit. Therefore, setting an extremely large value is unlikely to bring a significant increase in system performance. However, setting this parameter to hundreds of megabytes can improve the disk write performance on a server to which a large number of transactions are committed at the same time. The default value meets user requirements in most cases.

## wal\_writer\_delay

**Parameter description:** Specifies the delay between activity rounds for the WAL writer.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

A longer delay might lead to insufficient WAL buffer and a shorter delay leads to continuously writing of the WALs, thereby increasing the load of disk I/O.

---

**Value range:** an integer ranging from 1 to 10000. The unit is millisecond.

**Default value:** 200 ms

## commit\_delay

**Parameter description:** Specifies the duration for committed data to be stored in the WAL buffer.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- When this parameter is set to a non-zero value, the committed transaction is stored in the WAL buffer instead of being written to the WAL immediately. Then the WAL writer process flushes the buffer out to disks periodically.
  - If system load is high, other transactions are probably ready to be committed within the delay. If no other transactions are ready to be committed, the delay is a waste of time.
- 

**Value range:** an integer ranging from 0 to 100000. The unit is microsecond. **0** indicates no delay.

**Default value:** 0

## commit\_siblings

**Parameter description:** Specifies a threshold on the number of concurrent open transactions. If the number of concurrent open transactions is greater than the value of this parameter, a transaction that initiates a commit request will wait for a period of time specified by [commit\\_delay](#). Otherwise, this transaction is written into WALs immediately.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1000.

**Default value:** 5

## wal\_block\_size

**Parameter description:** Specifies the size of a WAL disk block.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Value range:** an integer. The unit is byte.

**Default value:** 8192

## wal\_segment\_size

**Parameter description:** Specifies the size of a WAL segment file.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Value range:** an integer. The unit is 8 KB.

**Default value:** 16 MB (2048 x 8 KB)

## force\_promote

**Parameter description:** Specifies whether to enable the forcible switchover function on the standby node.

When a cluster is faulty, the forcible switchover enables the cluster to recover services as soon as possible at the cost of losing some data. This is an escape method used when the cluster is unavailable. You are not advised to trigger this method frequently. You are not advised not to use this function if you are not clear about the impact of data loss on services.

To use this function, you need to enable it on the DN and CM Server and restart the cluster for the setting to take effect. For details about how to enable the forcible switchover function on the standby node, see "Emergency Handling > Performing a Forcible Primary/Standby Switchover" in *Troubleshooting*.

**Value range:** an integer. The value can be 0 (disabled) or 1 (enabled).

**Default value:** 0

## wal\_file\_init\_num

**Parameter description:** Specifies the number of Xlog segment files created by the WAL writer assistant thread at a time.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 1000000.

**Default value:** 10

## wal\_debug

**Parameter description:** Specifies whether to output WAL-related debugging information. This parameter is available only when **WAL\_DEBUG** is enabled during compilation.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**Default value:** false

## walwriter\_cpu\_bind

**Parameter description:** Sets the number of CPU cores bound to the WAL writer thread.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from -1 to 2147483647

**Default value:** -1

## walwriter\_sleep\_threshold

**Parameter description:** Specifies the number of times that the idle Xlog is refreshed before the Xlog refresher enters sleep.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 50000.

**Default value:** 500

## wal\_flush\_timeout

**Parameter description:** Specifies the timeout interval for traversing **WalInsertStatusEntryTbl**. It is the maximum wait time for the adaptive Xlog disk flushing I/O to traverse **WalInsertStatusEntryTbl**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

---

### NOTICE

If the timeout interval is too long, the Xlog flushing frequency may decrease, reducing the Xlog processing performance.

---

**Value range:** an integer ranging from 0 to 90000000. The unit is microsecond.

**Default value:** 2us

## wal\_flush\_delay

**Parameter description:** Specifies the wait interval when an entry in the **WAL\_NOT\_COPIED** state is encountered during **WalInsertStatusEntryTbl** traversal.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 90000000. The unit is microsecond.

**Default value:** 1us

## 14.3.6.2 Checkpoints

### checkpoint\_segments

**Parameter description:** Specifies the minimum number of WAL segment files in the period specified by [checkpoint\\_timeout](#). The size of each log file is 16 MB.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483646

Increasing the value of this parameter speeds up the import of a large amount of data. Set this parameter based on [checkpoint\\_timeout](#) and [shared\\_buffers](#). This parameter affects the number of WAL segment files that can be reused. Generally, the maximum number of reused files in the [pg\\_xlog](#) folder is twice the number of [checkpoint\\_segments](#). The reused files are not deleted and are renamed to the WAL segment files which will be later used.

**Default value:** 1024

### checkpoint\_timeout

**Parameter description:** Specifies the maximum time between automatic WAL checkpoints.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 30 to 3600. The unit is second.

If the value of [checkpoint\\_segments](#) is increased, you need to increase the value of this parameter. The increase of these two parameters further requires the increase of [shared\\_buffers](#). Consider all these parameters during setting.

**Default value:** 15min

### checkpoint\_completion\_target

**Parameter description:** Specifies the target of checkpoint completion.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a double-precision floating point number ranging from 0.0 to 1.0

**Default value:** 0.5

#### NOTE

0.5 indicates that each checkpoint should be complete within 50% of the interval between checkpoints.

### checkpoint\_warning

**Parameter description:** Specifies a time in seconds. If the checkpoint interval is close to this time due to filling of checkpoint segment files, a message is sent to the server log to suggest an increase in the value of [checkpoint\\_segments](#).

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is s. **0** indicates that the warning is disabled.

**Default value:** 5min

**Recommended value:** 5min

## checkpoint\_wait\_timeout

**Parameter description:** Sets the longest time that the checkpoint waits for the checkpoint thread to start.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 2 to 3600. The unit is s.

**Default value:** 1min

## enable\_incremental\_checkpoint

**Parameter description:** Specifies whether to enable incremental checkpoint.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**Default value:** on

## enable\_double\_write

**Parameter description:** Specifies whether to enable the doublewrite buffer. When the incremental checkpoint is enabled, the doublewrite buffer instead of **full\_page\_writes** is used to prevent partial page writes.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**Default value:** on

## incremental\_checkpoint\_timeout

**Parameter description:** Specifies the maximum interval between automatic WAL checkpoints when the incremental checkpoint is enabled.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 3600. The unit is second.

**Default value:** 1min

## enable\_xlog\_prune

**Parameter description:** Specifies whether the primary node reclaims logs if the size of Xlogs exceeds the value of **max\_size\_for\_xlog\_prune** when any standby node is disconnected.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- If this parameter is set to **on**, the primary node reclaims logs when any standby node is disconnected.
- If this parameter is set to **off**, the primary node does not reclaim logs when any standby node is disconnected.

**Default value:** on

## max\_size\_for\_xlog\_prune

**Parameter description:** This parameter takes effect when **enable\_xlog\_prune** is enabled. The working mechanism is as follows:

1. If all standby nodes specified by the **replconninfo** series GUC parameters are connected to the primary node, this parameter does not take effect.
2. If any standby node specified by the **replconninfo** series GUC parameters is not connected to the primary node, this parameter takes effect. When the number of historical logs on the primary node is greater than the value of this parameter, the logs are forcibly recycled. Exception: In synchronous commit mode (that is, the value of **synchronous\_commit** is not **local** or **off**), if there are connected standby nodes, the primary node retains the logs that meet the minimum log receiving requirements on the majority of standby nodes. In this case, the number of reserved logs may exceed the value of **max\_size\_for\_xlog\_prune**.
3. If any standby node is being built, this parameter does not take effect. All logs of the primary node are retained to prevent build failures due to log recycling.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is KB.

**Default value:** 256 GB

## max\_redo\_log\_size

**Parameter description:** On standby DN, this specifies the maximum size of logs between the latest checkpoint and the current log playback location. On the primary DN, this specifies the maximum size of logs between the recovery point and the latest log location. You are advised not to set this parameter to a large value if the RTO is concerned.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 163840 to 2147483647. The unit is KB.

**Default value:** 1048576. The unit is KB.

### 14.3.6.3 Log Replay

#### recovery\_time\_target

**Parameter description:** Sets the value of **recovery\_time\_target** in seconds to enable the standby server to write and replay logs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 3600. The unit is s.

0 indicates that log flow control is disabled. A value from 1 to 3600 indicates that a standby node can write and replay logs within the period specified by the value, so that the standby node can quickly assume the primary role. If this parameter is set to a small value, the performance of the primary server is affected. If it is set to a large value, the log flow is not effectively controlled.

**Default value:** 60

#### recovery\_max\_workers

**Parameter description:** Specifies the maximum number of concurrent replay threads.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 20.

**Default value:** 4

#### recovery\_parallelism

**Parameter description:** Specifies the actual number of replay threads. This parameter is read-only.

This parameter is a POSTMASTER parameter and is affected by **recovery\_max\_workers** and **recovery\_parse\_workers**. If any value is greater than 0, **recovery\_parallelism** will be recalculated.

**Value range:** an integer ranging from 1 to 2147483647.

**Default value:** 1

#### queue\_item\_size

**Parameter description:** Specifies the maximum length of the task queue of each redo replayer thread.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a value ranging from 1 to 65535.

**Default value:** 560

## recovery\_parse\_workers

**Parameter description:** Specifies the number of **ParseRedoRecord** threads for the ultimate RTO feature.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 16.

This parameter can be set to a value greater than 1 only when the ultimate RTO feature is enabled. In addition, it must be used together with **recovery\_redo\_workers**. If both **recovery\_parse\_workers** and **recovery\_max\_workers** are enabled, the ultimate RTO setting of **recovery\_parse\_workers** prevails and the concurrent replay function is disabled. In addition, to enable ultimate RTO, ensure that the value of [wal\\_receiver\\_buffer\\_size](#) is greater than or equal to 32 MB (64 MB is recommended).

**Default value:** 1

### NOTE

- After the database is upgraded from V500R001C00 to V500R001C10 or a later version, you are advised to set this parameter to **2** and restart the DN.
- After ultimate RTO is enabled, the standby node extra starts a number of threads (**recovery\_parse\_workers** × (**recovery\_redo\_workers** + 2) + 5), occupying more CPU, memory, and I/O resources. You need to select proper parameters. Otherwise, the standby node may fail to start due to insufficient resources.
- In this version and later, the ultimate RTO feature does not have flow control. Flow control is controlled by the [recovery\\_time\\_target](#) parameter.

## recovery\_redo\_workers

**Parameter description:** Specifies the number of **PageRedoWorker** threads corresponding to each **ParseRedoRecord** thread when the ultimate RTO feature is enabled.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 8.

It must be used together with **recovery\_parse\_workers**. The setting of **recovery\_redo\_workers** takes effect only when **recovery\_parse\_workers** is set to a value greater than 1.

**Default value:** 1

### NOTE

After the database is upgraded from V500R001C00 to V500R001C10 or a later version, you are advised to set parameters based on the number of CPUs in the environment and restart the DN.

**Table 14-5** Parameter settings for different CPUs, memory sizes, and deployment models

No.	CPUs	Memory	Distributed Hybrid Deployment	recover_parse_workers	recover_redo_workers	Replayer Threads	Remarks
1	4	-	-	1	1	-	Not recommended
2	8	-	Yes	1	1	-	Not recommended
3	8	64	No	1	1	-	Not recommended
4	16	128	Yes	1	1	-	Not recommended
5	16	128	No	2	3	15	-
6	32	256	Yes	2	2	13	-
7	32	256	No	2	8	25	-
8	64	512	Yes	2	4	17	-
9	64	512	No	2	8	25	Set the parameter to the recommended value for larger specifications.
10	96	768	-	2	8	25	Set the parameter to the recommended value for larger specifications.

## enable\_page\_lsn\_check

**Parameter description:** Specifies whether to enable the data page LSN check. During replay, the current LSN of the data page is checked to see if it is the expected one.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**Default value:** on

## redo\_bind\_cpu\_attr

**Parameter description:** Specifies the core binding operation of the replayer thread. Only user **sysadmin** can access this parameter. This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, consisting of one or more characters.

The available configuration modes are as follows: 1. '**nobind**': The thread is not bound to a core. 2. '**nodebind: 1, 2**': Use the CPU cores in NUMA groups 1 and 2 to bind threads. 3. '**cpubind: 0-30**': Use the CPU cores 0 to 30 to bind threads. The value of this parameter is case-insensitive.

**Default value:** '**nobind**'

 **NOTE**

This parameter is used for core binding in the Arm environment. You are advised to bind all replay threads to the same NUMA group for better performance.

## 14.3.6.4 Archiving

### archive\_mode

**Parameter description:** Specifies whether to archive WALs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

When [wal\\_level](#) is set to **minimal**, the **archive\_mode** parameter is unavailable.

---

**Value range:** Boolean

- **on** indicates that the archiving is enabled.
- **off** indicates that the archiving is disabled.

**Default value:** **off**

### archive\_command

**Parameter description:** Specifies the command set by the administrator to archive WALs. You are advised to set the archive log path to an absolute path.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

#### NOTICE

- If both **archive\_dest** and **archive\_command** are configured, WALs are preferentially saved to the directory specified by **archive\_dest**. The command configured by **archive\_command** does not take effect.
- Any **%p** in the string is replaced by the absolute path of the file to archive, and any **%f** is replaced by only the file name. (The relative path is relative to the data directory.) Use **%%** to embed an actual **%** character in the command.
- This command returns zero only if it succeeds. The command example is as follows:  

```
archive_command = 'cp --remove-destination %p /mnt/server/archivedir/%f'
```
- **--remove-destination** indicates that files will be overwritten during the archiving.
- If there are multiple archive commands, write them to the shell script file and set **archive\_command** to the command for executing the script. Example:  
-- Assume that multiple commands are as follows:  

```
test ! -f dir/%f && cp %p dir/%f
```

  
-- The content of the **test.sh** script is as follows:  

```
test ! -f dir/$2 && cp $1 dir/$2
```

  
-- The archive command is as follows:  

```
archive_command='sh dir/test.sh %p %f'
```

**Value range:** a string

**Default value:** (disabled)

## archive\_dest

**Parameter description:** Specifies the path set by the administrator to archive WALs. You are advised to set the archive log path to an absolute path.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

#### NOTICE

- If both **archive\_dest** and **archive\_command** are configured, WALs are preferentially saved to the directory specified by **archive\_dest**. The command configured by **archive\_command** does not take effect.
- If the string is a relative path, it is relative to the data directory. Example:  

```
archive_dest = '/mnt/server/archivedir/'
```

**Value range:** a string

**Default value:** an empty string

## archive\_timeout

**Parameter description:** Specifies the archiving period.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- The server is forced to switch to a new WAL segment file when the period specified by this parameter has elapsed since the last file switch.
- Archived files that are closed early due to a forced switch are still of the same length as full files. Therefore, a very short **archive\_timeout** will bloat the archive storage. You are advised to set **archive\_timeout** to **60s**.

---

**Value range:** an integer ranging from 0 to 1073741823. The unit is second. The value **0** indicates that the function is disabled.

**Default value:** 0

## archive\_interval

**Parameter description:** Specifies the archiving interval.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- Log files are forcibly archived when the period specified by this parameter has elapsed.
- Archiving involves I/O operations. Therefore, frequent archiving is not allowed. In addition, the RPO cannot be set to a large value; otherwise, the PITR will be affected. You are advised to use the default value.

---

**Value range:** an integer ranging from 1 to 1000. The unit is second.

**Default value:** 1

## time\_to\_target\_rpo

**Parameter description:** Specifies the maximum *time\_to\_target\_rpo* seconds from the time when an exception occurs on the primary cluster to the time when data is archived to the OBS recovery point in dual-cluster remote DR mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 3600. The unit is s.

In dual-cluster remote DR mode, logs of the primary cluster are archived to OBS. **0** indicates that log flow control is disabled. 1 to 3600 indicates the maximum *time\_to\_target\_rpo* seconds from the time when an exception occurs on the primary cluster to the time when data is archived to the recovery point of OBS. This ensures that the maximum duration of data loss is within the allowed range when the primary cluster breaks down due to a disaster. If this parameter is set to a small value, the performance of the primary server is affected. If it is set to a large value, the log flow is not effectively controlled.

**Default value:** 10

## 14.3.7 HA Replication

### 14.3.7.1 Sending Server

#### max\_wal\_senders

**Parameter description:** Specifies the maximum number of concurrent connections of transaction log sender processes. The value cannot be greater than or equal to that of [max\\_connections](#).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

#### NOTICE

[wal\\_level](#) must be set to **archive**, **hot\_standby**, or **logical** to allow connections from standby servers.

---

**Value range:** an integer ranging from 0 to 1024. The recommended value range is 8 to 100.

#### NOTE

This parameter can be set to 0 only when a single DN is used and there is no primary/standby instance.

#### **Default value:**

**Setting suggestions:** Each log replication connection between a standby server and a primary server occupies a WAL sender thread. Therefore, the value of this parameter must be greater than or equal to the number of DNs. Otherwise, the standby server cannot connect to the primary server. When logical replication is required, each log extraction thread occupies one WAL sender thread. If logical replication is required, set **max\_wal\_senders** to a value greater than the number of threads required by standby servers and logical replication extraction threads.

#### wal\_keep\_segments

**Parameter description:** Specifies the minimum number of transaction log files that can be retained in the **pg\_xlog** directory. The standby node obtains the logs from the primary node to perform streaming replication.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 2 to *INT\_MAX*

**Default value:** 128

#### **Setting suggestions:**

- During WAL archiving or recovery from a checkpoint on the server, the system may retain more log files than the number specified by **wal\_keep\_segments**.

- If this parameter is set to an excessively small value, a transaction log may have been overwritten by a new transaction before requested by the standby server. As a result, the request fails and the connection between the primary and standby servers is terminated.
- If the HA system uses asynchronous transmission, increase the value of **wal\_keep\_segments** when data greater than 4 GB is continuously imported in COPY mode. Take T6000 board as an example. If the data to be imported reaches 50 GB, you are advised to set this parameter to **1000**. You can dynamically restore the setting of this parameter after data import is complete and the log synchronization is normal.

## wal\_sender\_timeout

**Parameter description:** Specifies the maximum duration that the sender waits for the receiver to receive transaction logs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

- If the data volume on the primary node is huge, the value of this parameter must be increased for rebuilding. For example, if the data volume on the primary node reaches 500 GB, you are advised to set this parameter to 600 seconds.
- This parameter cannot be set to a value larger than the value of **wal\_receiver\_timeout** or the timeout parameter for database rebuilding.

---

**Value range:** an integer ranging from 0 to 2147483647. The unit is ms.

**Default value:** 6s

## max\_replication\_slots

**Parameter description:** Specifies the number of log replication slots in the primary server.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1024. The recommended value range is 8 to 100.

**Default value:** 20

**Setting suggestions:**

#### NOTICE

When HA replication, backup and restoration, and logical decoding are used, you are advised to set this parameter to: Number of current physical replication slots + Number of backup slots + Number of required logical replication slots.

If the actual value is smaller than the recommended value, these functions may be unavailable or abnormal.

Physical replication slots provide an automatic method to ensure that Xlog files are not removed from a primary DN before they are received by all the standby DNs, allowing high availability for the cluster. The number of physical replication slots required by the cluster is as follows: ratio of the number of standby DNs to the number of primary DNs in a group of DNs. For example, if the HA cluster has one primary DN and one standby DN, the number of physical replication slots required is 1. If the HA cluster has one primary DN and three standby DNs, the number of physical replication slots required is 3.

Backup slot records replication information during backup execution. Full backup and incremental backup correspond to two independent backup slots.

Plan the number of logical replication slots as follows:

- A logical replication slot can carry changes of only one database for decoding. If multiple databases are involved, create multiple logical replication slots.
- If logical replication is needed by multiple target databases, create multiple logical replication slots in the source database. Each logical replication slot corresponds to one logical replication link.

## max\_keep\_log\_seg

**Parameter description:** Stream control parameter. In logical replication, physical logs are parsed and converted into logical logs locally on the DN. When the number of physical log files that are not parsed is greater than the value of this parameter, stream control is triggered. The value **0** indicates that the stream control function is disabled.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 0

## enable\_wal\_shipping\_compression

**Parameter description:** Specifies whether to enable cross-cluster log compression in streaming DR mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- This parameter applies only to a pair of WAL senders and WAL receivers for cross-cluster transmission in streaming DR and is configured in the primary cluster.

---

**Value range:** Boolean

- **true** indicates that cross-cluster log compression is enabled in streaming DR mode.
- **false** indicates that cross-cluster log compression is disabled in streaming DR mode.

**Default value:** false

## repl\_auth\_mode

**Parameter description:** Specifies the validation mode for the primary/standby replication and standby node rebuilding.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- If UUID validation is enabled on the primary node and a non-null repl\_uuid validation code is configured, UUID validation must also be enabled on the standby node and the same repl\_uuid validation code must be configured on the standby node. Otherwise, requests for log replication between the primary and standby nodes and standby node rebuilding will be rejected by the primary node.
- The SIGHUP parameter can dynamically load new values. The modification does not affect the established primary/standby connection and takes effect for subsequent primary/standby replication requests and primary/standby rebuilding requests.
- It supports the standby node rebuild validation under the Quorum and DCF protocols and the primary/standby replication validation under the Quorum protocol. It does not support primary/standby replication validation under the DCF protocol.
- It does not support the authentication between the primary and standby nodes across clusters, including the primary/standby Dorado and DR clusters.
- The UUID validation function is used to prevent data crosstalk and pollution caused by incorrect connection between the primary and standby nodes. It is not used for security purposes.
- This parameter cannot be automatically synchronized between the primary and standby nodes.

---

**Value range:** enumerated values

- **off:** indicates that UUID validation is disabled.

- **default:** indicates that UUID validation is disabled.
- **uuid:** indicates that UUID validation is enabled.

**Default value:** default

## repl\_uuid

**Parameter description:** Specifies the UUID used for primary/standby UUID validation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

- If UUID validation is enabled on the primary node and a non-null repl\_uuid validation code is configured, UUID validation must also be enabled on the standby node and the same repl\_uuid validation code must be configured on the standby node. Otherwise, requests for log replication between the primary and standby nodes and standby node rebuilding will be rejected by the primary node.
- The SIGHUP parameter can dynamically load new values. The modification does not affect the established primary/standby connection and takes effect for subsequent primary/standby replication requests and primary/standby rebuilding requests.
- It supports the standby node rebuild validation under the Quorum and DCF protocols and the primary/standby replication validation under the Quorum protocol. It does not support primary/standby replication validation under the DCF protocol.
- It does not support the authentication between the primary and standby nodes across clusters, including the primary/standby Dorado and DR clusters.
- The UUID validation function is used to prevent data crosstalk and pollution caused by incorrect connection between the primary and standby nodes. It is not used for security purposes.
- This parameter cannot be automatically synchronized between the primary and standby nodes.

---

**Value range:** a string of 0 to 63 case-insensitive letters and digits. It is converted to lowercase letters for storage. An empty string indicates that UUID validation is disabled.

**Default value:** empty

## replconninfo1

**Parameter description:** Specifies the information about the first node to be listened on and authenticated by the current server. This parameter is automatically configured after the cluster is successfully installed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. An empty string indicates that no information about the first node is configured.

**Default value:** the first connection information listened on by the DN.

**Example:**

```
replconninfo1 = 'localhost= 127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost= 127.0.0.1 remoteservice=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

## replconninfo2

**Parameter description:** Specifies the information about the second node to be listened on and authenticated by the current server. This parameter is automatically configured after the cluster is successfully installed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. An empty string indicates that no information about the second node is configured.

**Default value:** information about the second connection listened to by the DN.

**Example:**

```
replconninfo2 = 'localhost= 127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost= 127.0.0.1 remoteservice=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

## replconninfo3

**Parameter description:** Specifies the information about the third node to be listened on and authenticated by the current server. This parameter is automatically configured after the cluster is successfully installed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. An empty string indicates that no information about the third node is configured.

**Default value:** information about the third connection listened to by the DN.

**Example:**

```
replconninfo3 = 'localhost= 127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost= 127.0.0.1 remoteservice=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

## replconninfo4

**Parameter description:** Specifies the information about the fourth node to be listened on and authenticated by the current server. This parameter is automatically configured after the cluster is successfully installed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. An empty string indicates that no information about the fourth node is configured.

**Default value:** information about the fourth connection listened to by the DN.

**Example:**

```
replconninfo4 = 'localhost= 127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost= 127.0.0.1 remoteservice=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

## replconninfo5

**Parameter description:** Specifies the information about the fifth node to be listened on and authenticated by the current server. This parameter is automatically configured after the cluster is successfully installed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string An empty string indicates that no information about the fifth node is configured.

**Default value:** information about the fifth connection listened to by the DN.

**Example:**

```
replconninfo5 = 'localhost= 127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost= 127.0.0.1 remoteservice=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

## replconninfo6

**Parameter description:** Specifies the information about the sixth node to be listened on and authenticated by the current server. This parameter is automatically configured after the cluster is successfully installed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. An empty string indicates that no information about the sixth node is configured.

**Default value:** information about the sixth connection listened to by the DN.

**Example:**

```
replconninfo6 = 'localhost= 127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost= 127.0.0.1 remoteservice=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

## replconninfo7

**Parameter description:** Specifies the information about the seventh node to be listened on and authenticated by the current server. This parameter is automatically configured after the cluster is successfully installed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. An empty string indicates that no information about the seventh node is configured.

**Default value:** information about the seventh connection listened to by the DN.

**Example:**

```
replconninfo7 = 'localhost= 127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost= 127.0.0.1 remoteservice=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

## enable\_time\_report

**Parameter description:** Specifies whether to record the time consumed by each redo log.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the time when a redo record is generated is recorded.
- **off** indicates that no record is generated.

**Default value:** off

## thread\_top\_level

**Parameter description:** Increases the priority of the WALWRITERAUXILIARY || WALWRITER || STARTUP ||WALRECEIVER || WAL\_NORMAL\_SENDER || PGSTAT threads to the highest.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the priority of the preceding threads is increased to the highest.
- **off** indicates that the priority of the preceding threads is not increased.

**Default value:** off

## wal\_flush\_size

**Parameter description:** Specifies the threshold for flushing entries in each WAL.

If this parameter is set to a large value for each entry, the flushing frequency is reduced, but the flushing delay increases.

The default value is **-1**, indicating that data is not evenly flushed and data is flushed as many as possible each time. If the parameter is set to a value greater than 0, data is evenly refreshed based on the threshold.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** [-1,16777216]. The unit is byte.

**Default value:** -1

## page\_work\_queue\_size

**Parameter description:** Specifies the length of the blocking queue of each redo worker.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a value ranging from 1 to 100000.

**Default value:** 4096

### 14.3.7.2 Primary Server

#### synchronous\_standby\_names

**Parameter description:** Specifies a comma-separated list of names of potential standby servers that support synchronous replication.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

#### NOTICE

- The current synchronous standby server is on the top of the list. If the current synchronous standby server is disconnected, it will be replaced immediately with the next-highest-priority standby server. Name of the next-highest-priority standby server is added to the list.
- The standby server name can be specified by setting the environment variable **PGAPPNAME**.

**Value range:** a string. If this parameter is set to \*, the name of any standby server that provides synchronous replication is matched. The value can be configured in the following format:

- ANY *num\_sync* (*standby\_name* [, ...])
- [FIRST] *num\_sync* (*standby\_name* [, ...])
- *standby\_name* [, ...]

#### NOTE

- In the preceding command, *num\_sync* indicates the number of standby nodes that need to wait for responses from the transaction, *standby\_name* indicates the name of the standby node, and FIRST and ANY specify the policies for selecting standby nodes for synchronous replication from the listed servers.
- **ANY N (dn\_instanceld1, dn\_instanceld2,...)** indicates that any *N* host names in the brackets are selected as the name list of standby nodes for synchronous replication. For example, **ANY 1 (dn\_instanceld1, dn\_instanceld2)** indicates that any one of **dn\_instanceld1** and **dn\_instanceld2** is used as the standby node for synchronous replication.
- **FIRST N (dn\_instanceld1, dn\_instanceld2, ...)** indicates that the first *N* primary node names in the brackets are selected as the standby node name list for synchronous replication based on the priority. For example, **FIRST 1 (dn\_instanceld1, dn\_instanceld2)** indicates that **dn\_instanceld1** is selected as the standby node for synchronous replication.
- The meanings of **dn\_instanceld1, dn\_instanceld2, ...** are the same as those of **FIRST 1 (dn\_instanceld1, dn\_instanceld2, ...)**.

If you use the `gs_guc` tool to set this parameter, perform the following operations:

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY NODE 1(dn_instanceld1, dn_instanceld2)'"
```

or

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY 1(AZ1, AZ2)'";
```

**Default value:** \*

## most\_available\_sync

**Parameter description:** Specifies whether to block the primary server when the primary-standby synchronization fails.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the primary server is not blocked when the synchronization fails.
- **off** indicates that the primary server is blocked when the synchronization fails.

**Default value:** off

## enable\_stream\_replication

**Parameter description:** Specifies whether data and logs are synchronized between primary and standby servers, and between primary and secondary servers.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

- This parameter is used for testing performance with standby DNs and without standby DNs. If this parameter is set to **off**, tests on abnormal scenarios, such as switchover and faults, cannot be performed to prevent inconsistency between the primary, standby, and secondary servers.
- This parameter is a controlled parameter, and you are advised not to set it to **off** in normal service scenarios.

---

**Value range:** Boolean

- **on** indicates that data and log synchronization is enabled.
- **off** indicates that data and log synchronization is disabled.

**Default value:** on

## enable\_mix\_replication

**Parameter description:** Specifies how WAL files and data are replicated between primary and standby servers, and between primary and secondary servers.

This parameter is an INTERNAL parameter. Its default value is **off** and cannot be modified.

---

**NOTICE**

This parameter cannot be modified in normal service scenarios. That is, the WAL file and data page mixed replication mode is disabled by default.

---

**Value range:** Boolean

- **on** indicates that the WAL file and data page mixed replication mode is enabled.
- **off** indicates that the WAL file and data page mixed replication mode is disabled.

**Default value:** off

### **vacuum\_defer\_cleanup\_age**

**Parameter description:** Specifies the number of transactions by which **VACUUM** will defer the cleanup of invalid row-store table records. That is, **VACUUM** and **VACUUM FULL** do not clean up deleted tuples immediately.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1000000. **0** means no delay.

**Default value:** 0

### **data\_replicate\_buffer\_size**

**Parameter description:** Specifies the amount of memory used by queues when the sender sends data pages to the receiver. The value of this parameter affects the buffer size used during the replication from the primary server to the standby server.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 4096 to 1072693248. The unit is KB.

**Default value:** 128 MB (131072 KB)

### **walsender\_max\_send\_size**

**Parameter description:** Specifies the size of the WAL or Sender buffers on the primary server.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 8 to 1048575. The unit is KB.

**Default value:** 8 MB (8192 KB)

## enable\_data\_replicate

**Parameter description:** Specifies how data is synchronized between primary and standby servers when the data is imported to a row-store table.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the primary and standby servers synchronize data using data pages when the data is imported to a row-store table. When **replication\_type** is set to **1**, this parameter cannot be set to **on**. If this parameter is set to **on** using the GUC tool, its value will be forcibly changed to **off**.
- **off** indicates that the primary and standby servers synchronize data using Xlogs when the data is imported to a row-store table.

**Default value:** off

## ha\_module\_debug

**Parameter description:** Specifies the replication status log of a specific data block during data replication.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the status of each data block is recorded in logs during data replication.
- **off** indicates that the status of each data block is not recorded in logs during data replication.

**Default value:** off

## enable\_incremental\_catchup

**Parameter description:** Specifies the data catchup mode between the primary and standby servers.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the standby server uses the incremental catchup mode. That is, the standby server scans local data files on the standby server to obtain the list of differential data files between the primary and standby servers and then performs catchup between the primary and standby servers.
- **off** indicates that the standby server uses the full catchup mode. That is, the standby server scans all local data files on the primary server to obtain the list of differential data files between the primary and standby servers and then performs catchup between the primary and standby servers.

**Default value:** on

## wait\_dummy\_time

**Parameter description:** Specifies the maximum duration for the primary server to wait for the standby servers to start and send the scanning lists when incremental data catchup is enabled in cluster.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647. The unit is second.

**Default value:** 300

### NOTE

The unit can only be second.

## catchup2normal\_wait\_time

**Parameter description:** Specifies the maximum duration that the primary server is blocked during the data catchup on the standby server in the case of a single synchronous standby server.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 10000. The unit is ms.

- The value **-1** indicates that the primary server is blocked until the data catchup on the standby server is complete.
- The value **0** indicates that the primary server is not blocked during the data catchup on the standby server.
- Other values indicate the maximum duration that the primary server is blocked during the data catchup on the standby server. For example, if this parameter is set to **5000**, the primary server is blocked until the data catchup on the standby server is complete in 5s.

**Default value:** -1

## sync\_config\_strategy

**Parameter description:** Specifies the policy for synchronizing configuration files between the primary node and standby node, and between the standby node and cascaded standby node.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **all\_node:** If this parameter is set to **all\_node** for the primary node, the primary node is allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **all\_node** for a standby node, the standby node is allowed to send synchronization requests to its primary node, and the standby node is allowed to proactively synchronize configuration files to all cascaded standby nodes. If this parameter is set to **all\_node** for a

cascaded standby node, the current cascaded standby node is allowed to send synchronization requests to its standby node.

- **only\_sync\_node**: If this parameter is set to **only\_sync\_node** for the primary node, the primary node is only allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **only\_sync\_node** for a standby node, the standby node is allowed to send synchronization requests to its primary node, and the standby node is not allowed to proactively synchronize configuration files to all cascaded standby nodes. If this parameter is set to **only\_sync\_node** for a cascaded standby node, the current cascaded standby node is allowed to send synchronization requests to its standby node.
- **none\_node**: If this parameter is set to **none\_node** for the primary node, the primary node is not allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **none\_node** for a standby node, the standby node is not allowed to send synchronization requests to its primary node, and the standby node is not allowed to proactively synchronize configuration files to all cascaded standby nodes. If this parameter is set to **none\_node** for a cascaded standby node, the current cascaded standby node is not allowed to send synchronization requests to its standby node.

**Default value:** all\_node

**Note:** If this parameter is configured after an upgrade from a version earlier than 503.0.0 to 503.0.0, this parameter cannot be identified after a rollback. Do not set this parameter during the upgrade to 503.0.0.

## hadr\_recovery\_time\_target

**Parameter description:** Specifies whether the standby database instance completes log writing and replay in streaming DR mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 3600. The unit is second.

0 indicates that log flow control is disabled. A value from 1 to 3600 indicates that a standby node can write and replay logs within the period specified by **hadr\_recovery\_time\_target**. This ensures that the logs can be written and replayed within the period specified by **hadr\_recovery\_time\_target** and the standby database instance can be promoted to primary quickly. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

**Default value:** 60 (financial edition (data computing))

## hadr\_recovery\_point\_target

**Parameter description:** Specifies the RPO time allowed for the standby database instance to flush logs to disks in streaming DR mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 3600. The unit is second.

**0** indicates that log flow control is disabled. A value from 1 to 3600 indicates that the standby node can flush logs to disks within the period specified by **hadr\_recovery\_point\_target**. This ensures that the log difference between the primary and standby database instances is controlled within the period specified by **hadr\_recovery\_point\_target** during the switchover and the standby database instance can be promoted to primary. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

**Default value:** 10 (financial edition (data computing))

## hadr\_super\_user\_record\_path

**Parameter description:** Specifies the path for storing encrypted files of the **hadr\_disaster** user in the standby cluster in streaming DR mode. This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Modification suggestion:** The value is automatically set by the streaming DR password transfer tool and does not need to be manually added.

**Value range:** a string

**Default value:** NULL

## check\_sync\_standby

**Parameter description:** Specifies whether to enable the standby node check function. After the **synchronous\_standby\_names** parameter is correctly configured in the primary/standby scenario, if the synchronous standby node is faulty, the write service on the primary node reports a write failure. This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** on or off

- **on** indicates that the standby node check is enabled.
- **off** indicates that the standby node check is disabled.

**Default value:** off

### NOTE

- This parameter cannot be synchronized in job work and autonomous transactions. Otherwise, the check may not take effect.
- If the standby node check is not configured for a specified user or session and the standby node is faulty when the forcible synchronization commit mode is enabled, the write operation on a table causes the query on the same table by another user or in another session to hang. In this case, you need to recover the standby node or manually terminate the hung client.
- The standby node check function cannot be enabled in scenarios (such as VACUUM ANALYZE and gs\_clean) where non-write operations trigger log writing. If the standby node does not meet the requirements for synchronizing configurations to the standby node, services will be hung in this scenario. In this case, you need to manually terminate the services.

### 14.3.7.3 Standby Server

#### hot\_standby

**Parameter description:** Specifies whether the standby server is allowed to accept connections and queries after it is restored to the minrecovery point.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

#### NOTICE

- If this parameter is set to **on**, [wal\\_level](#) must be set to **hot\_standby** or higher. Otherwise, the database startup fails.
- In a distributed system, **hot\_standby** cannot be set to **off**, because this setting can affect other features of the HA system.
- If the **hot\_standby** parameter was disabled and the **wal\_level** parameter was set to a value smaller than the value of **hot\_standby**, perform the following operations to ensure that the logs to be replayed on the standby node can be queried on the standby node before enabling the **hot\_standby** parameter again:
  1. Change the value of **wal\_level** of the primary and standby nodes to the value of **hot\_standby** or a higher value, and restart the instances for the change to take effect.
  2. Perform the checkpoint operation on the primary node and query the **pg\_stat\_get\_wal\_senders()** function to ensure that the value of **receiver\_replay\_location** of each standby node is the same as that of **sender\_flush\_location** of the primary node. Ensure that the value adjustment of **wal\_level** is synchronized to the standby nodes and takes effect, and the standby nodes do not need to replay low-level logs.
  3. Set the **hot\_standby** parameter of the primary and standby nodes to **on**, and restart the instances for the setting to take effect.

---

**Value range:** Boolean

- **on**: allowed.
- **off**: not allowed.

**Default value:** on

#### max\_standby\_archive\_delay

**Parameter description:** Specifies the wait period before queries on a standby server are canceled when the queries conflict with WAL processing and archiving in hot standby mode. In the current version, the setting does not take effect and is controlled by the **max\_standby\_streaming\_delay** parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

-1 indicates that the standby node waits until the conflicting queries are complete.

---

**Value range:** an integer ranging from -1 to 2147483647. The unit is ms.

**Default value:** 3s (3000 ms)

## max\_standby\_streaming\_delay

**Parameter description:** Specifies the wait period before queries on the standby node are canceled when the queries conflict with WAL data receiving through streaming replication in hot standby mode. If this parameter is set to a large value or the service load is heavy, an error may be reported for waiting for transaction replay and flushing to disks.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

-1 indicates that the standby node waits until the conflicting queries are complete.

---

**Value range:** an integer ranging from -1 to 2147483647. The unit is ms.

**Default value:** 3s (3000 ms)

## wal\_receiver\_status\_interval

**Parameter description:** Specifies the maximum interval for notifying the primary server of the WAL Receiver status.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

If this parameter is set to **0**, the standby server does not send information, such as the log receiving location, to the primary server. As a result, the transaction commit on the primary server may be blocked, and the switchover may fail. In normal service scenarios, you are advised not to set this parameter to **0**.

---

**Value range:** an integer ranging from 0 to 2147483. The unit is s.

**Default value:** 5s

## hot\_standby\_feedback

**Parameter description:** Specifies whether a standby server is allowed to send the result of a query performed on it to the primary server, preventing a query conflict.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the standby server is allowed to send the result of a query performed on it to the primary server.
- **off** indicates that the standby server is not allowed to send the result of a query performed on it to the primary server.

**Default value:** off

## wal\_receiver\_timeout

**Parameter description:** Specifies the maximum wait period for a standby server to receive data from the primary server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is ms.

**Default value:** 6s (6000 ms)

## wal\_receiver\_connect\_timeout

**Parameter description:** Specifies the timeout period for a standby server to connect to the primary server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483. The unit is s.

**Default value:** 2s

## wal\_receiver\_connect\_retries

**Parameter description:** Specifies the maximum attempts that a standby server connects to the primary server

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647

**Default value:** 1

## wal\_receiver\_buffer\_size

**Parameter description:** Specifies the memory buffer size for the standby and secondary nodes to store the received Xlog files.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 4096 to 1047552. The unit is KB.

**Default value:** 64 MB (65536 KB)

## primary\_slotname

**Parameter description:** Specifies the slot name of the primary server corresponding to a standby server. This parameter is used for the mechanisms to verify the primary-standby relationship and delete WALs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** empty

## enable\_redo\_atomic\_operation

**Parameter description:** Specifies whether to use atomic operations or spinlocks to update the LSN of the current thread when parallel playback is enabled.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that atomic operations are used for update.
- **off** indicates that spinlocks are used for update.

**Default value:** on

## max\_standby\_base\_page\_size

**Parameter description:** Specifies the maximum storage space of base page files on the standby node after the ultimate RTO function is enabled.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a long integer ranging from 1048576 to 562949953421311. The unit is KB.

**Default value:** 268435456 (256 GB)

## max\_standby\_lsn\_info\_size

**Parameter description:** Specifies the maximum size of LSN info files on the standby node after the ultimate RTO function is enabled.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a long integer ranging from 1048576 to 562949953421311. The unit is KB.

**Default value:** 268435456 (256 GB)

## base\_page\_saved\_interval

**Parameter description:** Specifies the interval for generating base pages on the standby node after the ultimate RTO function is enabled. For the same page, a base page is generated each time the value of this parameter is replayed.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 5 to 2000.

**Default value:** 400

## standby\_force\_recycle\_ratio

**Parameter description:** Specifies the percentage of files read by the standby node to trigger forcible recycling after the ultimate RTO function is enabled. When the total size of base page files exceeds the value of **max\_standby\_base\_page\_size** x **standby\_force\_recycle\_ratio** or the total size of LSN info files exceeds the value of **max\_standby\_lsn\_info\_size** x **standby\_force\_recycle\_ratio**, forcible recycling is triggered and some queries are canceled. When the value of **standby\_force\_recycle\_ratio** is **0**, forcible recycling is not started, and the setting of **max\_standby\_base\_page\_size** and **max\_standby\_lsn\_info\_size** does not take effect.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a double-precision floating point number ranging from 0.0 to 1.0

**Default value:** 0.8

## standby\_recycle\_interval

**Parameter description:** Specifies the interval for the standby node to recycle read files after the ultimate RTO function is enabled. The thread for recycling read resources on the standby node attempts to clear read files on the standby node at the interval specified by this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 86400. The unit is s.

**Default value:** 10

## standby\_max\_query\_time

**Parameter description:** Specifies the maximum query time supported on the standby node after the ultimate RTO function is enabled. If the query time exceeds the value of this parameter, the query will be canceled. Note: The time when the query is canceled is affected by the interval parameter [standby\\_recycle\\_interval](#) of the recycling thread and the time when the snapshot is obtained. Therefore, the actual execution time of the query on the standby node must be greater than the value of this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 86400. The unit is s.

**Default value:** 600

### exrto\_standby\_read\_opt

**Parameter description:** Specifies whether to support read optimization of the standby node with ultimate RTO. This parameter is enabled by default. This parameter is not synchronized between the primary and standby nodes.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean. **on** indicates that the optimization is enabled, and **off** indicates that the optimization is disabled.

**Default value:** on

## 14.3.8 Query Planning

This section describes the method configuration, cost constants, planning algorithm, and some configuration parameters for the optimizer.

### NOTE

- Two parameters are involved in the optimizer:
  - *INT\_MAX* indicates the maximum value of the INT data type. The value is **2147483647**.
  - *DBL\_MAX* indicates the maximum value of the FLOAT data type.
- In addition to customer services, global query planning parameters also affect database O&M and monitoring services, such as WDR generation, scale-out, redistribution, and data import and export.

### 14.3.8.1 Optimizer Method Configuration

These configuration parameters provide a crude method of influencing the query plans chosen by the query optimizer. If the default plan chosen by the optimizer for a particular query is not optimal, a temporary solution is to use one of these configuration parameters to force the optimizer to choose a different plan. Better ways include adjusting the optimizer cost constants, manually running **ANALYZE**, increasing the value of the **default\_statistics\_target** configuration parameter, and increasing the amount of the statistics collected in specific columns using **ALTER TABLE SET STATISTICS**.

### enable\_bitmapscan

**Parameter description:** Controls the query optimizer's use of bitmap-scan plan types.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on**: enabled.
- **off**: disabled.

**Default value:** on

## force\_bitmapand

**Parameter description:** Controls the query optimizer's use of BitmapAnd plan types.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on**: enabled.
- **off**: disabled.

**Default value:** off

## enable\_hashagg

**Parameter description:** Controls the query optimizer's use of Hash aggregation plan types.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on**: enabled.
- **off**: disabled.

**Default value:** on

## enable\_hashjoin

**Parameter description:** Controls the query optimizer's use of Hash-join plan types.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on**: enabled.
- **off**: disabled.

**Default value:** on

## enable\_indexscan

**Parameter description:** Controls the query optimizer's use of index-scan plan types.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** on

## enable\_indexonlyscan

**Parameter description:** Controls the query optimizer's use of index-only-scan plan types.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** on

## enable\_gsiscan

**Parameter description:** Controls the query optimizer's use of the global secondary index scan plan type. In the current version, global secondary indexes cannot be used for table access by index row ID.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** used.
- **off:** not used.

**Default value:** off

## enable\_material

**Parameter description:** Controls the query optimizer's use of materialization. It is impossible to suppress materialization entirely, but setting this variable to **off** prevents the optimizer from inserting materialized nodes.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** on

## enable\_mergejoin

**Parameter description:** Controls the query optimizer's use of merge-join plan types.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** off

## enable\_nestloop

**Parameter description:** Controls the query optimizer's use of nested-loop join plan types to fully scan internal tables. It is impossible to suppress nested-loop joins entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** off

## enable\_index\_nestloop

**Parameter description:** Controls the query optimizer's use of the index nested-loop join plan types to scan the parameterized indexes of internal tables.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** on

## enable\_seqscan

**Parameter description:** Controls the query optimizer's use of sequential scan plan types. It is impossible to suppress sequential scans entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on**: enabled.
- **off**: disabled.

**Default value:** on

## enable\_sort

**Parameter description:** Controls the query optimizer's use of sort methods. It is impossible to suppress explicit sorts entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on**: enabled.
- **off**: disabled.

**Default value:** on

## enable\_tidscan

**Parameter description:** Controls the query optimizer's use of Tuple ID (TID) scan plan types.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on**: enabled.
- **off**: disabled.

**Default value:** on

## enable\_kill\_query

**Parameter description:** In CASCADE mode, when a user is deleted, all the objects belonging to the user are deleted. This parameter specifies whether the queries of the objects belonging to the user can be unlocked when the user is deleted.

This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the unlocking is allowed.
- **off** indicates that the unlocking is not allowed.

**Default value:** off

## enable\_stream\_concurrent\_update

**Parameter description:** Controls the use of stream in concurrent updates. This parameter is restricted by the [enable\\_stream\\_operator](#) parameter. If concurrent

update conflicts occur, an error may be reported. This parameter can be used to control the conflict.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the optimizer can generate stream plans for the **UPDATE** statement.
- **off** indicates that the optimizer can generate only non-stream plans for the **UPDATE** statement.

**Default value:** on

## enable\_stream\_operator

**Parameter description:** Controls the query optimizer's use of stream. When this parameter is set to **off**, a large number of logs indicating that the stream plans cannot be pushed down are recorded. If you do not need these logs, you are advised to set both **enable\_unshipping\_log** and **enable\_stream\_operator** to **off**.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:**

- Independent deployment: **off**

## enable\_stream\_recursive

**Parameter description:** Specifies whether to push **WITH RECURSIVE** join queries to DNs for processing.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that **WITH RECURSIVE** join queries will be pushed down to DNs.
- **off** indicates that **WITH RECURSIVE** join queries will not be pushed down.

**Default value:** on

## max\_recursive\_times

**Parameter description:** Specifies the maximum number of **WITH RECURSIVE** iterations.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 200

## enable\_vector\_engine

**Parameter description:** Controls the query optimizer's use of vectorized executor.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** on

## enable\_broadcast

**Parameter description:** Controls the query optimizer's use of broadcast distribution method when it evaluates the cost of stream.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** on

## enable\_change\_hjcost

**Parameter description:** Specifies whether the optimizer excludes internal table running costs when selecting the Hash Join cost path. If it is set to **on**, tables with a few records and high running costs are more possible to be selected.

This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** off

## best\_agg\_plan

**Parameter description:** The query optimizer generates three plans for the aggregate operation under the stream:

1. hashagg+gather(redistribute)+hashagg
2. redistribute+hashagg(+gather)
3. hashagg+redistribute+hashagg(+gather)

This parameter is used to control the type of hashagg plans generated by the query optimizer.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 3.

- **1** indicates that the first plan is forcibly generated.
- **2** indicates that the second plan is forcibly generated if the **group by** column can be redistributed. Otherwise, the first plan is generated.
- **3** indicates that the third plan is forcibly generated if the **group by** column can be redistributed. Otherwise, the first plan is generated.
- **0** indicates that the optimizer chooses an optimal plan based on the estimated costs of the three plans above.

**Default value:** 0

## agg\_redistribute\_enhancement

**Parameter description:** When the aggregate operation is performed, which contains multiple **group by** columns and none of them is the distribution key, a **group by** column will be selected for redistribution. This parameter specifies the policy of selecting a redistribution column.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the column that can be redistributed and evaluates the most distinct value is selected for redistribution.
- **off** indicates that the first column that can be redistributed is selected for redistribution.

**Default value:** off

## enable\_absolute\_tablespace

**Parameter description:** Controls whether the tablespace can use an absolute path.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that an absolute path can be used.
- **off** indicates that an absolute path cannot be used.

**Default value:** on

## enable\_valuepartition\_pruning

**Parameter description:** Specifies whether the DFS partitioned table is dynamically or statically optimized.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the DFS partitioned table is dynamically or statically optimized.
- **off** indicates that the DFS partitioned table is not dynamically or statically optimized.

**Default value:** on

## expected\_computing\_nodegroup

**Parameter description:** Specifies a compute node group or the way to choose such a group. The node group mechanism is now for internal use only. You do not need to set it.

During join or aggregation operations, a node group can be selected in four modes. In each mode, the specified candidate compute node groups are listed for the optimizer to select the most appropriate one for the current operator.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string.

- **optimal:** The list of candidate compute node groups consists of the node groups where the operator's operation objects are located and the node group that combines all DNs in the node groups on which the current user has the COMPUTE permission.
- **query:** The list of candidate compute node groups consists of the node groups where the operator's operation objects are located and the node group that combines all DNs in the node groups where base tables involved in the query are located.
- *Node group name* (when [enable\\_nodegroup\\_debug](#) is set to **off**): The list of candidate compute node groups consists of the node groups where the operator's operation objects are located and the specified node group.
- *Node group name* (when [enable\\_nodegroup\\_debug](#) is set to **on**): A specific node group is used as the compute node group.

**Default value:** query

## enable\_nodegroup\_debug

**Parameter description:** Specifies whether the optimizer assigns computing workloads to a specific node group when multiple node groups exist in an environment. The node group mechanism is now for internal use only. You do not need to set it.

This parameter takes effect only when [expected\\_computing\\_nodegroup](#) is set to a specific node group.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that computing workloads are assigned to the node group specified by **expected\_computing\_nodegroup**.
- **off** indicates no node group is specified for computing.

**Default value:** off

## enable\_default\_index\_deduplication

**Parameter description:** Specifies whether to deduplicate and compress tuples with duplicate key values for a B-tree index by default. The deduplication and compression functions do not take effect for primary key indexes and unique indexes. When there are a large number of indexes with duplicate key values, the deduplication and compression function can effectively reduce the space occupied by indexes. In scenarios where non-unique indexes are used and index key values are seldom repeated or unique, the deduplication and compression function slightly deteriorates the index insertion performance. If the WITH (**deduplication** set to **on/off**) syntax is used during index creation, the deduplication parameter is preferentially used to determine whether to use deduplication and compression for the index.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **off:** indicates that the deduplication and compression function is disabled for B-tree indexes. This is the default value.
- **on:** indicates that the deduplication and compression function is enabled for B-tree indexes.

**Default value:** off

## enable\_expr\_fusion

**Parameter description:** Specifies whether to enable the SRF, expression flattening, centralized Seq Scan projection cancellation, transition status of shared aggregate functions, and step number optimization features.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **off:** indicates that this function is disabled. This is the default value.
- **on:** indicates that the SRF, expression flattening, centralized Seq Scan projection cancellation, transition status of shared aggregate functions, and step number optimization features are enabled.

**Default value:** off

### NOTE

It is supported only in the scenario where **query\_dop** is set to 1.

## stream\_multiple

**Parameter description:** Specifies the weight used by the optimizer to calculate the final cost of stream operators.

The base stream cost is multiplied by this weight to obtain the final cost.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

This parameter is applicable only to Redistribute and Broadcast streams.

---

**Value range:** a floating point number ranging from 0 to *DBL\_MAX*

**Default value:** 1

## qrw\_inlist2join\_optmode

**Parameter description:** Specifies whether to enable inlist-to-join (inlist2join) query rewriting.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string.

- **disable** indicates that the inlist2join query rewriting is disabled.
- **cost\_base** indicates that the cost-based inlist2join query rewriting is enabled.
- **rule\_base** indicates that the forcible rule-based inlist2join query rewriting is enabled.
- A positive integer indicates the threshold of inlist2join query rewriting. If the number of elements in the list is greater than the threshold, the rewriting is performed.

**Default value:** **cost\_base**

## skew\_option

**Parameter description:** Specifies whether an optimization policy is used.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **off** indicates that the policy is disabled.
- **normal** indicates that a radical policy is used. All possible skews are optimized.
- **lazy** indicates that a conservative policy is used. Uncertain skews are ignored.

**Default value:** **normal**

## enable\_dngather

**Parameter description:** Specifies whether to calculate stream plans that meet the threshold on a single DN to reduce the number of planned stream nodes.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** off

## dngather\_min\_rows

**Parameter description:** Specifies the maximum number of rows that control **dngather**. Values less than or equal to this parameter value can be calculated on a single DN. The prerequisite is that **enable\_dngather** is enabled.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from -1 to *DBL\_MAX*

**Default value:** 500.0

## cost\_weight\_index

**Parameter description:** Specifies the cost weight of index\_scan.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 1e-10 to 1e+10.

**Default value:** 1

## default\_limit\_rows

**Parameter description:** Specifies the default estimated number of limit rows for generating genericplan. If this parameter is set to a positive value, the positive value is used as the estimated number of limit rows. If this parameter is set to a negative value, the negative value is converted to a percentage and used as default estimated value, that is, -5 indicates 5%.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from -100 to *DBL\_MAX*

**Default value:** -10

## enforce\_a\_behavior

**Parameter description:** Controls the rule matching modes of regular expressions.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the A matching rule is used.
- **off** indicates that the POSIX matching rule is used.

**Default value:** on

## enable\_force\_vector\_engine

**Parameter description:** Specifies whether to forcibly generate vectorized execution plans for a vectorized execution operator if the operator's child node is a non-vectorized operator. When this parameter is set to **on**, vectorized execution plans are forcibly generated.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that vectorized operators are forcibly generated.
- **off** indicates that the vectorized operator optimizer determines whether to perform vectorization.

**Default value:** off

## check\_implicit\_conversions

**Parameter description:** Specifies whether to check candidate index paths generated for index columns that have implicit type conversions in a query. For details about the application scenarios of this parameter, see "SQL Optimization > Checking the Implicit Conversion Performance" in *Developer Guide*.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that a check will be performed for candidate index paths generated for index columns that have implicit type conversion in a query.
- **off** indicates that a check will not be performed.

**Default value:** off

---

### NOTICE

When this parameter is set to **on**, you need to set **enable\_fast\_query\_shipping** to **off** so that the mechanism for identifying implicit data type conversion of index columns can take effect.

---

## 14.3.8.2 Optimizer Cost Constants

This section describes the optimizer cost constants. The cost variables described here are measured on an arbitrary scale. Only their relative values matter,

therefore scaling them all up or down by the same factor will result in no change in the optimizer's choices. By default, these cost variables are based on the cost of sequential page fetches, that is, **seq\_page\_cost** is conventionally set to **1.0** and the other cost variables are set with reference to the parameter. However, you can use a different scale, such as actual execution time in milliseconds.

## seq\_page\_cost

**Parameter description:** Specifies the optimizer's estimated cost of a disk page fetch that is part of a series of sequential fetches.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0 to *DBL\_MAX*

**Default value:** 1

## random\_page\_cost

**Parameter description:** Specifies the optimizer's estimated cost of an out-of-sequence disk page fetch.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

Although the server allows you to set **random\_page\_cost** to a value less than that of **seq\_page\_cost**, it is not physically sensitive to do so. However, setting them equal makes sense if the database is entirely cached in RAM, because in that case there is no penalty for fetching pages out of sequence. Also, in a heavily-cached database you should lower both values relative to the CPU parameters, since the cost of fetching a page already in RAM is much smaller than it would normally be.

---

**Value range:** a floating point number ranging from 0 to *DBL\_MAX*

**Default value:** 4

### NOTE

- This value can be overwritten for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name.
- Reducing this value relative to **seq\_page\_cost** will cause the system to prefer index scans and raising it will make index scans relatively more expensive. You can increase or decrease both values together to change the disk I/O costs relative to CPU costs.

## cpu\_tuple\_cost

**Parameter description:** Specifies the optimizer's estimated cost of processing each row during a query.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0 to *DBL\_MAX*

**Default value:** 0.01

## cpu\_index\_tuple\_cost

**Parameter description:** Specifies the optimizer's estimated cost of processing each index entry during an index scan.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0 to *DBL\_MAX*

**Default value:** 0.005

## cpu\_operator\_cost

**Parameter description:** Specifies the optimizer's estimated cost of processing each operator or function executed during a query.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0 to *DBL\_MAX*

**Default value:** 0.0025

## effective\_cache\_size

**Parameter description:** Specifies the optimizer's assumption about the effective size of the disk cache that is available to a single query.

Set this parameter based on the following factors: the GaussDB's shared buffer space, the kernel's disk buffer space, and the estimated number of concurrent queries on different tables that share the available space.

This parameter does not affect the size of the shared memory allocated during actual GaussDB running. It is used only for estimation in the plan generation phase. The value is in the unit of disk page. Usually the size of each page is 8192 bytes.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647. The unit is 8 KB.

**Default value:**

Independent deployment:

CN: **2 GB** (60-core CPU/480 GB memory); **1 GB** (32-core CPU/256 GB memory and 16-core CPU/128 GB memory); **512 MB** (8-core CPU/64 GB memory); **256 MB** (4-core CPU/32 GB memory); **128 MB** (4-core CPU/16 GB memory)

DN: **70 GB** (60-core CPU/480 GB memory); **38 GB** (32-core CPU/256 GB memory); **20 GB** (16-core CPU/128 GB memory); **8 GB** (8-core CPU/64 GB memory); **4 GB** (4-core CPU/32 GB memory); **2 GB** (4-core CPU/16 GB memory)

Setting suggestions:

A larger value indicates that the optimizer prefers index scanning, and a smaller value indicates that the optimizer prefers full table scanning. Generally, the value is half of the value of **shared\_buffers**. More radically, you can set the value to three-fourth of the value of **shared\_buffers**.

## allocate\_mem\_cost

**Parameter description:** Specifies the query optimizer's estimated cost of creating a hash table for memory space using hash join. This parameter is used for optimization when the hash join estimation is inaccurate.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0 to *DBL\_MAX*

**Default value:** 0

### 14.3.8.3 Genetic Query Optimizer

This section describes parameters related to genetic query optimizer. The genetic query optimizer (GEQO) is an algorithm that plans queries by using heuristic searching. This algorithm reduces planning time for complex queries and the costs of producing plans are sometimes inferior to those found by the normal exhaustive-search algorithm.

## geqo

**Parameter description:** Specifies whether to enable the genetic query optimization.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

#### NOTICE

It is best not to turn it off in execution. **geqo\_threshold** provides more subtle control of GEQO.

If this parameter is modified by running the **gs\_guc reload** command and the connection of a session on the current node is not from the client but from another node in the cluster to which the node belongs, this parameter does not take effect immediately on the session after the **gs\_guc reload** command is executed. The setting takes effect only after the connection node is disconnected and then reconnected.

---

**Value range:** Boolean

- **on** indicates that the genetic query optimization is enabled.
- **off** indicates that the genetic query optimization is disabled.

**Default value:** on

## geqo\_threshold

**Parameter description:** Specifies the number of **FROM** items. Genetic query optimization is used to plan queries when the number of statements executed is greater than this value.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

- For simpler queries it is best to use the regular, exhaustive-search planner, but for queries with many tables it is better to use GEQO to manage the queries.
  - A **FULL OUTER JOIN** construct counts as only one **FROM** item.
- 

**Value range:** an integer ranging from 2 to 2147483647

**Default value:** 12

## geqo\_effort

**Parameter description:** Controls the trade-off between planning time and query plan quality in GEQO.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

**geqo\_effort** does not do anything directly. This parameter is only used to compute the default values for the other variables that influence GEQO behavior. If you prefer, you can manually set the other parameters instead.

---

**Value range:** an integer ranging from 1 to 10

---

### NOTICE

Larger values increase the time spent in query planning, but also increase the probability that an efficient query plan is chosen.

---

**Default value:** 5

## geqo\_pool\_size

**Parameter description:** Controls the pool size used by GEQO, that is, the number of individuals in the genetic population.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

---

**NOTICE**

The value of this parameter must be at least **2**, and useful values are typically from **100** to **1000**. If this parameter is set to **0**, GaussDB selects a proper value based on **geqo\_effort** and the number of tables.

---

**Default value:** 0

## geqo\_generations

**Parameter description:** Specifies the number of iterations of the GEQO.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

---

**NOTICE**

The value of this parameter must be at least **1**, and useful values are typically from **100** to **1000**. If it is set to **0**, a suitable value is chosen based on **geqo\_pool\_size**.

---

**Default value:** 0

## geqo\_selection\_bias

**Parameter description:** Specifies the selection bias used by GEQO. The selection bias is the selective pressure within the population.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 1.5 to 2.0

**Default value:** 2

## geqo\_seed

**Parameter description:** Specifies the initial value of the random number generator used by GEQO to select random paths through the join order search space.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0.0 to 1.0

---

**NOTICE**

Varying the value changes the set of join paths explored, and may result in a better or worse best path being found.

---

**Default value:** 0

### 14.3.8.4 Other Optimizer Options

#### cost\_model\_version

**Parameter description:** Specifies the version of the optimizer cost model. It can be regarded as a protection parameter to disable the latest optimizer cost model and keep consistent with the plan of the earlier version. Changing the value of this parameter may change many SQL plans. Therefore, exercise caution when changing the value of this parameter.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0, 1, 2, or 3

- **0** indicates that the latest cost estimation model is used. The current version is equivalent to **3**.
- **1** indicates that the original cost estimation model is used.
- **2:** indicates that the enhanced COALESCE expression, hash join cost, and semi/anti join cost are used for estimation on the basis of **1**.
- **3:** indicates that the boundary correction estimator is used to estimate the NDV on the basis of **2**. The hint of indexscan can be applied to indexonlyscan.

**Default value:** 0

#### enable\_fast\_query\_shipping

**Parameter description:** Specifies whether to use the distributed framework for a query optimizer.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the distributed framework is not used. Execution plans are generated on CNs and DNs separately.
- **off** indicates that the distributed framework is used. Execution plans are generated on CNs and then sent to DN for execution.

**Default value:** on

#### enable\_trigger\_shipping

**Parameter description:** Specifies whether the trigger can be pushed to DN for execution.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the trigger can be pushed to DN for execution.

- **off** indicates that the trigger cannot be pushed to DNs. It must be executed on CNs.

**Default value:** on

## enable\_remotejoin

**Parameter description:** Specifies whether JOIN operation plans can be delivered to DNs for execution.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that JOIN operation plans can be delivered to DNs for execution.
- **off** indicates that JOIN operation plans cannot be delivered to DNs for execution.

**Default value:** on

## enable\_remotegroup

**Parameter description:** Specifies whether the execution plans of **GROUP BY** and **AGGREGATE** can be delivered to DNs for execution.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the execution plans of **GROUP BY** and **AGGREGATE** can be delivered to DNs for execution.
- **off** indicates that the execution plans of **GROUP BY** and **AGGREGATE** cannot be delivered to DNs for execution.

**Default value:** on

## enable\_remotelimit

**Parameter description:** Specifies whether the execution plan specified in the LIMIT clause can be delivered to DNs for execution.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the execution plan specified in the LIMIT clause can be pushed down to DNs for execution.
- **off** indicates that the execution plan specified in the LIMIT clause cannot be delivered to DNs for execution.

**Default value:** on

## enable\_remosort

**Parameter description:** Specifies whether the execution plan of the **ORDER BY** clause can be delivered to DNs for execution.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the execution plan of the **ORDER BY** clause can be delivered to DNs for execution.
- **off** indicates that the execution plan of the **ORDER BY** clause cannot be delivered to DNs for execution.

**Default value:** on

## enable\_csqual\_pushdown

**Parameter description:** Specifies whether to deliver filter criteria for a rough check during query.

This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that a rough check is performed with filter criteria delivered during query.
- **off** indicates that a rough check is performed without filter criteria delivered during query.

**Default value:** on

## explain\_dna\_file

**Parameter description:** Sets [explain\\_perf\\_mode](#) to **run** to export object files in CSV format.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

The value of this parameter must be an absolute path plus a file name with the extension **.csv**.

---

**Value range:** a string

**Default value:** empty

## analysis\_options

**Parameter description:** Specifies whether to enable function options in the corresponding options to use the corresponding location functions, including data

verification and performance statistics. For details, see the options in the value range.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

- **HASH\_CONFLICT** indicates that the log file in the **pg\_log** directory of the DN process displays the hash table statistics, including the hash table size, hash chain length, and hash conflict information.
- **STREAM\_DATA\_CHECK** indicates that a CRC check is performed on data before and after network data transmission.

**Default value:** ALL,on(),off(HASH\_CONFLICT,STREAM\_DATA\_CHECK), which indicates that no location function is enabled.

## explain\_perf\_mode

**Parameter description:** Specifies the display format of the **explain** command.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** normal, pretty, summary, and run

- **normal** indicates that the default printing format is used.
- **pretty** indicates a new format improved by using GaussDB. The new format contains a plan node ID, directly and effectively analyzing performance.
- **summary** indicates that the analysis result on this information is printed in addition to the printed information specified by **pretty**.
- **run** indicates that the system exports the printed information specified by **summary** as a CSV file for further analysis.

**Default value:** pretty

### NOTE

The pretty mode supports only plans that contain stream operators and does not support plans that deliver statements to DNs. Therefore, the display format is affected by the **enable\_stream\_operator** parameter. When **enable\_stream\_operator** is set to **off**, the plan containing the stream operator cannot be generated.

## cost\_param

**Parameter description:** Controls use of different estimation methods in specific customer scenarios, allowing estimated values approximating to onsite values. This parameter can control various methods simultaneously by performing AND (&) on the bit of each method. A method is selected if the result value is not 0.

- When **cost\_param & 1** is set to a value other than 0, an improved mechanism is used for connecting the selection rate of non-equi-joins. This method is more accurate for estimating the selection rate of joins between two identical tables. At present, if **cost\_param & 1** is set to a value other than 0, the path is not used. That is, a better formula is selected for calculation.
- When **cost\_param & 2** is set to a value other than 0, the selection rate is estimated based on multiple filter criteria. The lowest selection rate among all

filter criteria, but not the product of the selection rates for two tables under a specific filter criterion, is used as the total selection rate. This method is more accurate when a close correlation exists between the columns to be filtered.

- When **cost\_param & 4** is not 0, the selected debugging model is not recommended when the stream node is evaluated.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 0

## enable\_partitionwise

**Parameter description:** Specifies whether to select an intelligent algorithm for joining partition tables.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that an intelligent algorithm is selected.
- **off** indicates that an intelligent algorithm is not selected.

**Default value:** off

## enable\_fast\_numeric

**Parameter description:** Specifies whether to enable optimization for numeric data calculation. Calculation of numeric data is time-consuming. Numeric data is converted into int64- or int128-type data to improve numeric data calculation performance.

This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** or **true** indicates that optimization for numeric data calculation is enabled.
- **off** or **false** indicates that optimization for numeric data calculation is disabled.

**Default value:** on

## rewrite\_rule

**Parameter description:** Specifies the rewriting rule for enabled optional queries. Some query rewrite rules are optional. Enabling them cannot always improve the query efficiency. In a specific customer scenario, you can set the query rewriting rules through this GUC parameter to achieve optimal query efficiency.

This parameter can control the combination of query rewriting rules, for example, there are multiple rewriting rules: rule1, rule2, rule3, and rule4. You can perform the following settings:

```
set rewrite_rule=rule1; -- Enable query rewriting rule rule1
set rewrite_rule=rule2, rule3; -- Enable the query rewriting rules rule2 and rule3
set rewrite_rule=none; -- Disable all optional query rewriting rules
```

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

- **none:** Does not use any optional query rewriting rules
- **Lazyagg:** Uses the Lazy Agg query rewriting rules for eliminating aggregation operations in subqueries
- **magicset:** The Magic Set query rewriting rules are used to associate subqueries which have aggregation operators with the main query in advance to reduce repeated scanning of sublinks.
- **partialpush:** Uses the Partial Push query rewriting rules. For statements that cannot be pushed down, push down some subqueries to DN for execution and the rest to CN for execution.
- **uniquecheck:** Uses the Unique Check query rewriting rules. Optimize the subquery statements in target columns without agg and check whether the number of returned rows is 1.
- **disablerep:** Uses the Disable Replicate query rewriting rules. The performance may deteriorate after a replication table is optimized. Therefore, after this rule is enabled, subqueries cannot be optimized.
- **intargetlist:** Uses the In Target List query rewriting rules (subquery optimization in the target column).
- **predpushnormal:** Uses the Predicate Push query rewriting rules. When predicate conditions are pushed down to subqueries, the BROADCAST operator may be added to support distributed execution.
- **predpushforce:** Uses the Predicate Push query rewriting rules. Push down predicate conditions to subqueries and use indexes as much as possible for acceleration.
- **predpush:** Selects the optimal plan based on the cost in **predpushnormal** and **predpushforce**.
- **disable\_pullup\_expr\_sublink:** The optimizer is not allowed to pull up sublinks of the expr\_sublink type. For details about sublink classification and pull-up principles, see "SQL Optimization > Typical SQL Optimization Methods > Optimizing Subqueries" in *Developer Guide*.
- **enable\_sublink\_pullup\_enhanced:** Enhanced sublink query rewriting rules are used, including unrelated sublink pull-up of the WHERE and HAVING clauses and WinMagic rewriting optimization.
- **disable\_pullup\_not\_in\_sublink:** The optimizer is not allowed to pull up sublinks related to NOT IN. For details about sublink classification and pull-up principles, see "SQL Optimization > Typical SQL Optimization Methods > Optimizing Subqueries" in *Developer Guide*.

**Default value:** magicset

## enable\_pbe\_optimization

**Parameter description:** Specifies whether the optimizer optimizes the query plan for statements executed in Parse Bind Execute (PBE) mode.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the optimizer optimizes the query plan for statements executed in PBE mode.
- **off** indicates that the optimizer does not optimize the execution.

**Default value:** on

## enable\_light\_proxy

**Parameter description:** Specifies whether the optimizer optimizes the execution of simple queries on CNs. This parameter does not take effect if the character set of the application side does not match that of the kernel side. You are advised to set the character set to UTF8 when creating a database.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the optimizer optimizes the execution of simple queries on CNs.
- **off** indicates that the optimizer does not optimize the execution.

**Default value:** on

## enable\_global\_plancache

**Parameter description:** Specifies whether to share the cache for the execution plans of statements in PBE queries and stored procedures. If this parameter is set to **on**, the memory usage of the CNs and DN in high concurrency scenarios can be reduced. In addition, the value of this parameter must be the same on the CN and DN. Otherwise, the packets sent from the CN to the DN do not match and an error is reported.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

When **enable\_global\_plancache** is enabled, the default value of **local\_syscache\_threshold** is greater than or equal to 16 MB to ensure that GPC takes effect. If the value of **local\_syscache\_threshold** is less than 16 MB, set it to 16 MB. If the value is greater than 16 MB, do not change it.

**Value range:** Boolean

- **on** indicates that cache sharing is enabled for the execution plans of statements in PBE queries and stored procedures.
- **off** indicates no sharing.

**Default value:** off

## gpc\_clean\_timeout

**Parameter description:** When **enable\_global\_plancache** is set to **on**, if a plan in the shared plan list is not used within the period specified by **gpc\_clean\_timeout**,

the plan will be deleted. This parameter is used to control the retention period of a shared plan that is not used.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 300 to 86400. The unit is s.

**Default value:** 1800, that is, 30 minutes

## enable\_gpc\_grayrelease\_mode

**Parameter description:** Specifies whether to enable GPC in a distributed cluster. The cluster needs to be restarted to enable GPC. If you want to enable GPC without restarting the cluster, use **enable\_gpc\_grayrelease\_mode**.

Operations in a distributed cluster:

To enable GPC:

1. Enable **enable\_gpc\_grayrelease\_mode** on all DNs.
2. Enable **enable\_gpc\_grayrelease\_mode** on all CNs.
3. Enable the GPC parameter which is a POSTMASTER parameter. You need to reload the parameter and then kill the node in polling mode for GPC on the restarted node to take effect.

To disable GPC:

1. Ensure that **enable\_gpc\_grayrelease\_mode** is set to **on**, reload and then disable the GPC parameter, and kill the node in polling mode for GPC on the restarted node to take effect.
2. Disable **enable\_gpc\_grayrelease\_mode** on all CNs.
3. Disable **enable\_gpc\_grayrelease\_mode** on all DNs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- on
- off

**Default value:** off

## enable\_opfusion

**Parameter description:** Specifies whether to optimize simple queries.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

## NOTICE

This parameter is used to optimize the query performance of DNs. You can set **max\_datanode\_for\_plan** to view the execution plan of a query DN. If the execution plan of the DN contains **[Bypass]**, the query can be optimized on the DN.

The restrictions on simple queries are as follows:

- Only indexscan and indexonlyscan are supported, and the filter criteria of all **WHERE** statements are on indexes.
- Only single tables can be added, deleted, modified, and queried. JOIN and USING operations are not supported.
- Only row-store tables are supported. Partitioned tables and tables with triggers are not supported.
- Information statistics features of active SQL statements and queries per second (QPS) are not supported.
- Tables that are being scaled out or in are not supported.
- System columns cannot be queried or modified.
- Only simple **SELECT** statements are supported. For example:  

```
SELECT c3 FROM t1 WHERE c1 = ? and c2 =10;
```

Only columns in the target table can be queried. Columns **c1** and **c2** are index columns, which can be followed by constants or parameters. You can use **for update**.

- Only simple **INSERT** statements are supported. For example:  

```
INSERT INTO t1 VALUES (?,10,?);
```

Only one **VALUES** is supported. The type in **VALUES** can be a constant or a parameter. **RETURNING** is not supported.

- Only simple **DELETE** statements are supported. For example:  

```
DELETE FROM t1 WHERE c1 = ? and c2 = 10;
```

Columns **c1** and **c2** are index columns, which can be followed by constants or parameters.

- Only simple **UPDATE** statements are supported. For example:  

```
UPDATE t1 SET c3 = c3+? WHERE c1 = ? and c2 = 10;
```

The values modified in column **c3** can be constants, parameters, or a simple expression. Columns **c1** and **c2** are index columns, which can be followed by constants or parameters.

**Value range:** Boolean

- **on** indicates that the performance logs are output.
- **off** indicates that the performance logs are not output.

**Default value:** on

## enable\_partition\_opfusion

**Parameter description:** If this parameter is enabled when the enable\_opfusion parameter is enabled, the simple query of the partitioned table can be optimized to improve the SQL execution performance. If **enable\_global\_plancache** is set to **on**, this parameter does not take effect.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the performance logs are output.
- **off** indicates that the performance logs are not output.

**Default value:** off

## sql\_beta\_feature

**Parameter description:** Specifies the SQL engine's optional beta features to be enabled, including optimization of row count estimation and query equivalence estimation. These optional features provide optimization for specific scenarios, but performance deterioration may occur in some scenarios for which testing is not performed. In a specific customer scenario, you can set the query rewriting rules through this GUC parameter to achieve optimal query efficiency.

This parameter determines the combination of the SQL engine's beta features, for example, feature1, feature2, feature3, and feature4. You can perform the following settings:

```
set sql_beta_feature=feature1; --Enable the beta feature 1 of the SQL engine.
set sql_beta_feature=feature2,feature3; --Enable the beta features 2 and 3 of the SQL engine.
set sql_beta_feature=none; --Disable all optional SQL engine beta features.
```

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

- **none:** None of the beta optimizer features are used.
- **sel\_semi\_poisson:** Uses poisson distribution to calibrate the equivalent semi-join and anti-join selection rates.
- **sel\_expr\_instr:** Uses the matching row count estimation method to provide more accurate estimation for **instr(col, 'const') > 0, = 0, = 1**.
- **param\_path\_gen:** Generates more possible parameterized paths.
- **rand\_cost\_opt:** Optimizes the random read cost of tables that have a small amount of data.
- **param\_path\_opt:** Uses the bloating ratio of the table to optimize the analysis information of indexes.
- **page\_est\_opt:** Optimizes the **relpages** estimation for the analysis information of table indexes.
- **no\_unique\_index\_first:** Disables optimization of the primary key index scanning path first.
- **join\_sel\_with\_cast\_func:** Supports type conversion functions when the number of join rows is estimated.
- **canonical\_pathkey:** The regularization pathkey is generated in advance (**pathkey**: a set of ordered key values of data).

 **WARNING**

After this parameter is enabled, the semantics of the output data of statements such as ORDER BY may be different from that of the standard ones in the case of outer join. Contact Huawei engineers to determine whether to enable this parameter.

- **index\_cost\_with\_leaf\_pages\_only**: specifies whether index leaf nodes are included when the index cost is estimated.
- **partition\_opfusion**: enables partitioned table optimization.
- **a\_style\_coerce**: enables the Decode type conversion rule to be compatible with O. For details, see the part related to case processing in ORA compatibility mode in "SQL Reference > Type Conversion > UNION, CASE, and Related Constructs" in *Developer Guide*.
- **plpgsql\_stream\_fetchall**: enables the function of obtaining all tuple results when the SQL statements which use streams are executed on the for loop or cursor in a stored procedure.
- **partition\_fdw\_on**: SQL statements can be created for Postgres foreign tables based on partitioned tables.
- **predpush\_same\_level**: enables the **predpush** hint to control parameterized paths at the same layer.
- **disable\_bitmap\_cost\_with\_lossy\_pages**: disables the computation of the cost of lossy pages in the bitmap path cost.
- **enable\_upsert\_execute\_gplan**: In the PBE scenario, if the UPDATE clause in the on duplicate key update statement contains parameters, set **enable\_upsert\_execute\_gplan** to allow execution through **gplan**.

**Default value:**

"sel\_semi\_poisson,sel\_expr\_instr,rand\_cost\_opt,param\_path\_opt,page\_est\_opt"

## table\_skewness\_warning\_threshold

**Parameter description:** Specifies the threshold for triggering a table skew alarm.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0 to 1

**Default value:** 1

## table\_skewness\_warning\_rows

**Parameter description:** Specifies the minimum number of rows for triggering a table skew alarm.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 100000

## enable\_global\_stats

**Parameter description:** Specifies the current statistics collection mode, which can be global statistics collection or single-node statistics collection. By default, the global statistics collection mode is used. If this parameter is disabled, the statistics of the first node in the cluster are collected by default. In this case, the quality of the generated query plan may be affected. However, the information collection performance is optimal. Therefore, exercise caution when disabling this parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** or **true** indicates the global statistics mode.
- **off** or **false** indicates the single-DN statistics mode.

**Default value:** on

## default\_statistics\_target

**Parameter description:** Specifies the default statistics target for table columns without a column-specific target set via **ALTER TABLE SET STATISTICS**. If this parameter is set to a positive number, it indicates the number of samples of statistics information. If this parameter is set to a negative number, percentage is used to set the statistic target. The negative number converts to its corresponding percentage, for example, -5 means 5%.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -100 to 10000

---

### NOTICE

- A larger positive number than the default value increases the time required to do **ANALYZE**, but might improve the quality of the optimizer's estimates.
- Changing settings of this parameter may result in performance deterioration. If query performance deteriorates, you can:
  1. Restore to the default statistics.
  2. Use hints to force the optimizer to use the optimal query plan. For details, see "SQL Optimization > Hint-based Tuning" in *Developer Guide*.
- If this GUC parameter is set to a negative value, the number of samples is greater than or equal to 2% of the total data volume, and the number of records in user tables is less than 1.6 million, the time taken by running **ANALYZE** will be longer than that when this parameter uses its default value.
- If this GUC parameter is set to a negative value, the autoanalyze function is disabled.

---

**Default value:** 100

## default\_gsi\_statistics\_target

**Parameter description:** Sets the default statistics target for global secondary indexes. If the parameter is set to a positive number, it indicates the number of samples of statistics information. If the parameter is set to a negative number, it indicates the percentage of statistics collected. The negative number converts to its corresponding percentage, for example, **-5** means 5%.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer, in the range from -100 to 3000000.

**Default value:** 0

---

### NOTICE

- The default value **0** indicates that the default sampling rate is used for sampling.
  - Changing settings of this parameter may result in performance deterioration. If query performance deteriorates, you can:
    1. Restore to the default statistics.
    2. Use hints to force the optimizer to use the optimal query plan. For details, see "SQL Optimization > Hint-based Tuning" in *Developer Guide*.
- 

## auto\_statistic\_ext\_columns

**Parameter description:** Collects statistics about multiple columns based on the first *K* columns of the composite index in the data table. This GUC parameter indicates *K*. For example, if a composite index is (a,b,c,d,e) and the GUC parameter is set to **3**, statistics about multiple columns are generated on columns (a,b) and (a,b,c). Multi-column statistics can make the optimizer estimate the cardinality more accurate when querying with combined conditions.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- The system catalog does not take effect.
- The statistics take effect only when the types of all columns support the comparison functions '=' and '<'.
- System pseudocolumns in indexes, such as **tableoid** and **ctid**, are not collected.
- By default, distinct values, MCVs without NULL, and MCVs with NULL are collected. If the AI-based cardinality estimation parameter **enable\_ai\_stats** is set to **on**, MCVs are not collected. Instead, models for AI-based cardinality estimation are collected.
- If the index for creating multi-column statistics is deleted and no other index contains the multi-column combination, the multi-column statistics will be deleted in the next ANALYZE operation.
- If the value of this parameter decreases, the new index generates multi-column statistics based on the value of this parameter. The generated multi-column statistics that exceed the value of this parameter will not be deleted.
- If you want to disable the multi-column statistics on a specific combination only, you can retain the value of this parameter and run the **ALTER TABLE tablename disable statistics ((column list))** DDL command to disable the statistics on multiple columns in a specific combination.

---

**Value range:** an integer ranging from 1 to 4 The value 1 indicates that statistics about multiple columns are not automatically collected.

**Default value:** 1

## constraint\_exclusion

**Parameter description:** Controls the query optimizer's use of table constraints to optimize queries.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **on, true, yes,** and **1** indicate that constraints for all tables are examined.
- **off, false, no,** and **0** indicate that no constraints are examined.
- **partition** indicates that only constraints for inheritance child tables and **UNION ALL** subqueries are examined.

---

**NOTICE**

When **constraint\_exclusion** is set to **on**, the optimizer compares query conditions with the table's **CHECK** constraints, and omits scanning tables for which the conditions contradict the constraints.

---

**Default value:** partition

 NOTE

Currently, **constraint\_exclusion** is enabled by default only for cases that are often used to implement table partitioning. Turning this feature on for all tables imposes extra planning on simple queries, and provides no benefit for simple queries. If you have no partitioned tables, set it to **off**.

## cursor\_tuple\_fraction

**Parameter description:** Specifies the optimizer's estimated fraction of a cursor's rows that are retrieved.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0.0 to 1.0

---

**NOTICE**

Smaller values of this setting bias the optimizer towards using **fast start** plans for cursors, which will retrieve the first few rows quickly while perhaps taking a long time to fetch all rows. Larger values put more emphasis on the total estimated time. At the maximum setting of **1.0**, cursors are planned exactly like regular queries, considering only the total estimated time and how soon the first rows might be delivered.

---

**Default value:** 0.1

## from\_collapse\_limit

**Parameter description:** Specifies whether the optimizer merges sub-queries into upper queries based on the resulting FROM list. The optimizer merges sub-queries into upper queries if the resulting FROM list would have no more than this many items.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647

---

**NOTICE**

Smaller values reduce planning time but may lead to inferior execution plans.

---

**Default value:** 8

## join\_collapse\_limit

**Parameter description:** Specifies whether the optimizer rewrites **JOIN** constructs (except **FULL JOINS**) into lists of **FROM** items based on the number of the items in the result list.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647

---

**NOTICE**

- Setting this parameter to **1** prevents join reordering. As a result, the join order specified in the query will be the actual order in which the relations are joined. The query optimizer does not always choose the optimal join order. Therefore, advanced users can temporarily set this variable to **1**, and then specify the join order they desire explicitly.
- Smaller values reduce planning time but lead to inferior execution plans.

---

**Default value:** 8

## plan\_mode\_seed

**Parameter description:** This is a commissioning parameter. Currently, it supports only **OPTIMIZE\_PLAN** and **RANDOM\_PLAN**. The value **0** (for **OPTIMIZE\_PLAN**) indicates the optimized plan using the dynamic planning algorithm. Other values are for **RANDOM\_PLAN**, which indicates that the plan is randomly generated. **-1** indicates that users do not specify the value of the seed identifier. In this case, the optimizer generates a random integer from **1** to **2147483647** and a random execution plan based on the generated integer. A GUC parameter value from **1** to **2147483647** is regarded as the seed identifier, based on which the optimizer generates a random execution plan.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 2147483647

**Default value:** 0

---

**NOTICE**

- If this parameter is set to **RANDOM\_PLAN**, the optimizer generates a random execution plan that may not be the optimal one. Therefore, to guarantee the query performance, the default value **0** is recommended during upgrade, scale-out, scale-in, and O&M.
- If this parameter is not set to **0**, the specified plan hint will not be used.

## enable\_random\_datanode

**Parameter description:** Specifies whether the query of the replication table is conducted on a random DN. A complete replication table is stored on each each DN for random retrieval to release the pressure on nodes.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the function of querying the replication table on a random DN is enabled.
- **off** indicates that the function of querying the replication table on a random DN is disabled.

**Default value:** on

## hashagg\_table\_size

**Parameter description:** Specifies the hash table size during the execution of the HASH JOIN operation.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1073741823

**Default value:** 0

## enable\_bloom\_filter

**Parameter description:** Specifies whether the BloomFilter optimization is used. This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the BloomFilter optimization can be used.
- **off** indicates that the BloomFilter optimization cannot be used.

**Default value:** on

## enable\_extrapolation\_stats

**Parameter description:** Specifies whether the extrapolation logic is used for data of DATE type based on historical statistics. The logic can increase the accuracy of estimation for tables whose statistics are not collected in time, but will possibly provide an overlarge estimation due to incorrect extrapolation. Enable the logic only in scenarios where the data of DATE type is periodically inserted. This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the extrapolation logic is used for data of DATE type based on historical statistics.
- **off** indicates that the extrapolation logic is not used for data of DATE type based on historical statistics.

**Default value:** off

## autoanalyze

**Parameter description:** Specifies whether to automatically collect statistics on tables that have no statistics when a plan is generated. **autoanalyze** cannot be

used for foreign or temporary tables. To collect statistics, manually perform the ANALYZE operation. If an exception occurs in the database during the execution of autoanalyze on a table, after the database is recovered, the system may still prompt you to collect the statistics of the table when you run the statement again. In this case, manually perform the ANALYZE operation on the table to synchronize statistics. This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the table statistics are automatically collected.
- **off** indicates that the table statistics are not automatically collected.

**Default value:** off

## query\_dop

**Parameter description:** Specifies the user-defined degree of parallelism (DOP). This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -64 to 64

A value ranging from 1 to 64 indicates that the fixed SMP is enabled and the system will use the specified DOP.

**0** indicates that the SMP adaptation is enabled, and the system will dynamically select the optimal DOP based on resource usage and plan characteristics.

A value ranging from -64 to -1 indicates that the SMP adaptation is enabled, and the system limits the DOP that can be adaptively selected.

### NOTE

- After enabling concurrent queries, ensure you have sufficient CPU, memory, network, and I/O resources to achieve the optimal performance.
- To prevent performance deterioration caused by an overly large value of **query\_dop**, the system calculates the maximum number of available CPU cores for a DN and uses the number as the upper limit for this parameter. If the value of **query\_dop** is greater than 4 and also the upper limit, the system resets **query\_dop** to the upper limit.

**Default value:** 1

## enable\_analyze\_check

**Parameter description:** Checks whether statistics were collected about tables whose **reltuples** and **relpages** are displayed as **0** in **pg\_class** during plan generation.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the tables will be checked.
- **off** indicates that the tables will not be checked.

**Default value:** off

## enable\_sonic\_hashagg

**Parameter description:** Specifies whether to use the hash aggregation operator designed for column-oriented hash tables when certain constraints are met.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the hash aggregation operator designed for column-oriented hash tables is used when certain constraints are met.
- **off** indicates that the hash aggregation operator designed for column-oriented hash tables is not used.

### NOTE

- If **enable\_sonic\_hashagg** is enabled and the Hash Agg operator designed based on the column-oriented hash table is used when the query meets the constraint condition, the memory usage of the Hash Agg operator can be reduced. However, in scenarios where the performance is significantly improved based on code generation, the performance of the operator may deteriorate.
- If **enable\_sonic\_hashagg** is enabled and the hash agg operator designed based on the column-oriented hash table is used when the query meets the constraint condition, the operator is displayed as Sonic Hash Aggregation in the execution plan and execution information of Explain Analyze/Performance; when the query does not meet the constraint condition, the operator is displayed as Hash Aggregation. For details, see "SQL Optimization > Introduction to the SQL Execution Plan > Description" in *Developer Guide*.

**Default value:** on

## enable\_sonic\_hashjoin

**Parameter description:** Specifies whether to use the hash join operator designed for column-oriented hash tables when certain constraints are met.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the hash join operator designed for column-oriented hash tables is used when certain constraints are met.
- **off** indicates that the hash join operator designed for column-oriented hash tables is not used.

 NOTE

- Currently, the parameter can be used only for Inner Join.
- If **enable\_sonic\_hashjoin** is enabled, the memory usage of query using the Hash Inner operator can be reduced. However, in scenarios where the code generation technology can significantly improve performance, the performance of the operator may deteriorate.
- If **enable\_sonic\_hashjoin** is enabled and the hash join operator designed based on the column-oriented hash table is used when the query meets the constraint condition, the operator is displayed as Sonic Hash Join in the execution plan and execution information of Explain Analyze/Performance; when the query does not meet the constraint condition, the operator is displayed as Hash Join. For details, see "SQL Optimization > Introduction to the SQL Execution Plan > Description" in *Developer Guide*.

**Default value:** on

## enable\_sonic\_optspill

**Parameter description:** Specifies whether to optimize the number of files to be written to disks for the Hash Join operator designed for column-oriented hash tables. If this parameter is set to **on**, the number of files written to disks does not increase significantly when the Hash Join operator writes a large number of files to disks.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the optimization is enabled.
- **off** indicates that the optimization is disabled.

**Default value:** on

## plan\_cache\_mode

**Parameter description:** Specifies the policy for generating an execution plan in the **prepare** statement.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **auto** indicates that the **custom plan** or **generic plan** is selected by default.
- **force\_generic\_plan** indicates that the **generic plan** (soft parse) is forcibly used. The **generic plan** is a plan generated after you run a prepared statement. The plan policy binds parameters to the plan when you run the EXECUTE statement and execute the plan. The advantage of this plan is that repeated optimizer overheads can be avoided in each execution. The disadvantage is that the plan may not be optimal when data skew occurs for the bound parameters and may result in poor plan execution performance.
- **force\_custom\_plan** indicates that the **custom plan** (hard parse) is forcibly used. The **custom plan** is a plan generated after you run a prepared statement where parameters in the EXECUTE statement are embedded. The **custom plan** generates a plan based on specific parameters in the EXECUTE

statement. This plan generates a preferred plan based on specific parameters each time and has good execution performance. The disadvantage is that the plan needs to be regenerated before each execution, resulting in a large amount of repeated optimizer overhead.

 **NOTE**

This parameter is valid only for prepared statements. It is used when the parameterized field in a prepared statement has severe data skew.

**Default value:** auto

## enable\_router

**Parameter description:** Specifies whether to enable the manual node pushdown function.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the performance logs are output.
- **off** indicates that the performance logs are not output.

**Default value:** off

## router

**Parameter description:** Controls the detailed attributes of the router function. This parameter is valid only when **enable\_router** and **enable\_light\_proxy** are enabled. This parameter is used to calculate the DN where the given distribution key is located based on the hash distribution key of the table. After the router is set, the supported SQL statements are pushed down to the DN for execution. If the router is incorrectly configured, data may be saved to an incorrect DN, causing unpredictable problems. Therefore, be cautious when running this command.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

This parameter consists of two parts:

'**schema\_name.table\_name,"distribute\_keys"**'. The meanings are as follows:

- **schema\_name.table\_name:** indicates the schema name and table name. If schema\_name is not set, the default value current\_schema is used.
- **distribute\_keys:** Values of all distribution keys in the distribution table are separated by commas (.). The sequence of the values must be the same as that of the distribution keys in the table.

**Default value:** empty

## enable\_auto\_explain

**Parameter description:** Specifies whether to enable the function of automatically printing execution plans. This parameter is used to locate slow stored procedures or slow queries and is valid only for the currently connected CN.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean. The value **on** indicates that the function is enabled, and the value **off** indicates that the function is disabled.

**Default value:** off

## auto\_explain\_level

**Parameter description:** Specifies the log level for automatically printing execution plans.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Enumeration type. The value can be **log** or **notice**. **log** indicates that the execution plan is printed in logs. **notice** indicates that the execution plan is printed in notification mode.

**Default value:** log

## auto\_explain\_log\_min\_duration

**Parameter description:** Specifies the minimum duration of execution plans that are automatically printed. Only execution plans whose duration is greater than the value of **auto\_explain\_log\_min\_duration** will be printed.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is ms.

- **0:** All executed plans are printed.
- **3000:** All execution plans will be printed if the execution of a statement takes more than 3000 ms.

**Default value:** 0

## max\_datanode\_for\_plan

**Parameter description:** Specifies the number of execution plans to be displayed on the DN when an FQS plan is generated. The number of plans that are displayed on the DN is determined by the smaller value between the number of DNs on the cluster and the value of this parameter.

For statements executed by PBE, only plans generated in kernel prepared precompilation mode can be displayed. Plans generated in JDBC precompilation mode cannot be displayed.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 8192

**Default value:** 0

## enable\_inner\_unique\_opt

**Parameter description:** Specifies that Inner Unique is optimized for nested loop join, hash join, and sort merge join. That is, the number of matching times is reduced when the attribute corresponding to the inner table in the join condition meets the uniqueness constraint.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** used.
- **off:** not used.

**Default value:** on

## enable\_indexscan\_optimization

**Parameter description:** Specifies whether to optimize B-tree index scanning (IndexScan and IndexOnlyScan) in the Astore engine.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** used.
- **off:** not used.

**Default value:** off

## session\_sequence\_cache

**Parameter description:** Specifies the **sequence** value applied for one-time interaction in the current session. The unused values are automatically discarded after the session ends. When using **sequence** to import data in batches, you can increase the value of this parameter to improve the insertion speed and high concurrency performance. When a single data record is inserted concurrently, set this parameter to **1** to reduce the sequence change. If you have high requirements on continuity, you need to specify the required cache when creating a sequence. If the value of this parameter is greater than that of cache, the value automatically becomes invalid.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647

**Default value:** 10

### NOTE

The default value is **10**. In high-concurrency scenarios, the performance of single and batch insertion is good.

## enable\_dynamic\_sample\_size

**Parameter description:** Specifies whether to dynamically adjust the number of sampled rows. For a large table with more than one million rows, the number of sampled rows is dynamically adjusted during statistics collection to improve statistics accuracy.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** indicates that this function is enabled.
- **off:** indicates that the function is disabled.

**Default value:** on

### NOTE

The function of dynamically adjusting the number of sampled rows supports only absolute sampling.

## 14.3.9 Error Reporting and Logging

### 14.3.9.1 Logging Destination

#### log\_destination

**Parameter description:** GaussDB supports several methods of logging server messages. Set this parameter to a list of desired log destinations separated by commas. (For example, log\_destination="stderr, csvlog")

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string.

The valid values are **stderr**, **csvlog**, and **syslog**.

- **stderr** indicates that logs are printed to the screen.
- **csvlog** indicates that logs are output in comma-separated value (CSV) format. The prerequisite for generating logs in CSV format is that **logging\_collector** must be set to **on**. For details, see [Using CSV Log Output](#).
- **syslog** indicates that logs are recorded using the syslog of the OS. GaussDB can record logs using syslog from **LOCAL0** to **LOCAL7**. For details, see [syslog facility](#). To record logs using syslog, add the following information to syslog daemon's configuration file:

```
local0.* /var/log/postgresql
```

**Default value:** stderr

#### logging\_collector

**Parameter description:** Specifies whether to enable the logger process to collect logs. This process captures log messages sent to **stderr** or **csvlog** and redirects them into log files.

This method is more effective than recording logs to syslog because some types of messages cannot be displayed in syslog output, such as messages indicating the loading failures of dynamic link libraries and error messages generated by scripts (for example, **archive\_command**).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

It is possible to log to **stderr** without using the logging collector and the log messages will go to where the server's **stderr** is directed. However, this method is only suitable for low log volumes due to difficulties in rotating log files.

---

**Value range:** Boolean

- **on** indicates that the log collection is enabled.
- **off** indicates that the log collection is disabled.

**Default value:** on

## log\_directory

**Parameter description:** Specifies the directory for storing log files when **logging\_collector** is set to **on**. The value can be an absolute path, or relative to the data directory. The **log\_directory** parameter can be dynamically modified using the **gs\_guc reload** command. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

- If this parameter is set to an invalid path, the cluster cannot be started.
  - If you modify the **log\_directory** parameter using the **gs\_guc reload** command, and the specified path is valid, the log files are output to this new path. If the specified path is invalid, the log files are output to the valid path set last time and the database operation is not affected. The invalid value is still written into the configuration file.
  - In the sandbox environment, the path cannot contain `/var/chroot`. For example, if the absolute path of log is `/var/chroot/var/lib/log/Ruby/pg_log/cn_log`, you only need to set the path to `/var/lib/log/Ruby/pg_log/cn_log`.
- 

**NOTE**

- Valid path: Users have read and write permissions on the path.
- Invalid path: Users do not have read or write permissions on an invalid path.

**Value range:** a string

**Default value:** specified during installation

## log\_filename

**Parameter description:** Specifies the names of generated log files when **logging\_collector** is set to **on**. The value is treated as a strftime pattern, so %-escapes can be used to specify time-varying file names. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

- You are advised to use %-escapes to specify the log file names for efficient management of log files.
- If **log\_destination** is set to **csvlog**, log files are output in CSV format with timestamped names, for example, **server\_log.1093827753.csv**.

---

**Value range:** a string

**Default value:** postgresql-%Y-%m-%d\_%H%M%S.log

## log\_file\_mode

**Parameter description:** Specifies the permissions of log files when **logging\_collector** is set to **on**. This parameter is invalid on Windows. The parameter value is usually a number in the format acceptable to the **chmod** and **umask** system calls.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

- Before setting this parameter, set **log\_directory** to store the logs to a directory other than the data directory.
- Do not make the log files world-readable because they might contain sensitive data.

---

**Value range:** an octal integer ranging from 0000 to 0777 (that is, 0 to 511 in the decimal format)

### NOTE

- **0600** indicates that log files are readable and writable only to the server administrator.
- **0640** indicates that log files are readable and writable to members of the administrator's group.

**Default value:** 0600

## log\_truncate\_on\_rotation

**Parameter description:** Specifies the writing mode of the log files when **logging\_collector** is set to **on**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

A setting example is as follows:

Assume that you want logs to be kept for 7 days, a log file generated each day to be named **server\_log.Mon** on Monday, **server\_log.Tue** on Tuesday, and so forth, and this week's log files to be overwritten by next week's log files. Then you can set **log\_filename** to **server\_log.%a**, **log\_truncate\_on\_rotation** to **on**, and **log\_rotation\_age** to **1440** (indicating that the valid duration of the log file is 24 hours).

**Value range:** Boolean

- **on** indicates that GaussDB overwrites the existing log files of the same name on the server.
- **off** indicates that GaussDB appends the logging messages to the existing log files of the same name on the server.

**Default value:** **off**

## log\_rotation\_age

**Parameter description:** Specifies the interval for creating a log file when **logging\_collector** is set to **on**. If the duration from the time when the last log file was created to the current time is greater than the value of **log\_rotation\_age**, a new log file will be generated.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 35791394. The unit is min. **0** indicates that the time-based creation of new log files is disabled.

**Default value:** **1d** (1440 min)

## log\_rotation\_size

**Parameter description:** Specifies the maximum size of a server log file when **logging\_collector** is set to **on**. If the total size of messages in a log file exceeds the specified value, a log file will be generated.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2097151. The unit is KB.

**0** indicates that the capacity-based creation of new log files is disabled.

**Default value:** 20 MB

## syslog\_facility

**Parameter description:** Specifies the syslog facility to be used when **log\_destination** is set to **syslog**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values. Valid values are **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6**, and **local7**.

**Default value:** local0

## syslog\_ident

**Parameter description:** Specifies the identifier of GaussDB messages in syslog logs when **log\_destination** is set to **syslog**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** postgres

## event\_source

**Parameter description:** Specifies the identifier of GaussDB messages in logs when **log\_destination** is set to **eventlog**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** PostgreSQL

## 14.3.9.2 Logging Time

### client\_min\_messages

**Parameter description:** Specifies which level of messages will be sent to the client. Each level covers all the levels following it. The lower the level is, the fewer messages are sent.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

A same value for **client\_min\_messages** and **log\_min\_messages** does not indicate the same level.

---

**Value range:** enumerated type. The valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal** and **panic**.

Among them, **debug** and **debug2** are equivalent. For details about the parameters, see [Table 14-6](#). If the configured level is higher than **error**, for example, **fatal** or **panic**, the system changes the level to **error** by default.

**Default value:** notice

## log\_min\_messages

**Parameter description:** Specifies which level of messages will be written into the server log. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

A same value for **client\_min\_messages** and **log\_min\_messages** does not indicate the same level.

---

**Value range:** enumerated type. The valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal** and **panic**. Among them, **debug** and **debug2** are equivalent. For details about the parameters, see [Table 14-6](#).

**Default value:** warning

## log\_min\_error\_statement

**Parameter description:** Controls which SQL statements that cause an error condition are recorded in the server log.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameters, see [Table 14-6](#).

### NOTE

- The default is **error**, indicating that statements causing errors, log messages, fatal errors, or panics will be logged.
- **panic** indicates that SQL statements that cause an error condition will not be logged.

**Default value:** error

## log\_min\_duration\_statement

**Parameter description:** Specifies the threshold for logging the duration of a completed statement. If a statement runs for a duration greater than or equal to the specified value, its duration will be logged.

Setting this parameter can be helpful in tracking down unoptimized queries. For clients using extended query protocols, the time required for parsing, binding, and executing steps are logged independently.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

When this option is used together with [log\\_statement](#), the text of statements that are logged because of [log\\_statement](#) will not be repeated in the duration log message. If you are not using [syslog](#), it is recommended that you log the process ID (PID) or session ID using [log\\_line\\_prefix](#) so that you can link the statement message to the later duration message.

---

**Value range:** an integer ranging from -1 to 2147483647. The unit is ms.

- If this parameter is set to **250**, all SQL statements that run for 250 ms or longer will be logged.
- **0** indicates that the execution durations of all the statements are logged.
- **-1** indicates that the duration logging is disabled.

**Default value:** 3s (that is, 3000 ms)

## backtrace\_min\_messages

**Parameter description:** Prints the function's stack information to the server's log file if the information generated is greater than or equal to the level specified by this parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

This parameter is used to locate problems on-site. Frequent stack printing will affect the system's overhead and stability. Therefore, set the value of this parameter to a rank other than **fatal** or **panic** during problem location.

---

**Value range:** enumerated values

Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameters, see [Table 14-6](#).

**Default value:** panic

[Table 14-6](#) explains message severities used by GaussDB. If logging output is sent to syslog or eventlog, severity is translated in GaussDB as shown in the table.

**Table 14-6** Message severity levels

Severity	Description	System Log	Event Log
debug[1-5]	Provides detailed debug information.	DEBUG	INFORMATION
log	Reports information of interest to administrators, for example, checkpoint activity.	INFO	INFORMATION
info	Provides information implicitly requested by users, for example, output from <b>VACUUM VERBOSE</b> .	INFO	INFORMATION
notice	Provides information that might be helpful to users, for example, truncation of long identifiers and index created as part of the primary key.	NOTICE	INFORMATION
warning	Provides warnings of likely problems, for example, <b>COMMIT</b> outside a transaction block.	NOTICE	WARNING
error	Reports an error that causes a command to terminate.	WARNING	ERROR
fatal	Reports the reason that causes a session to terminate.	ERR	ERROR
panic	Reports an error that caused all database sessions to terminate.	CRIT	ERROR

### 14.3.9.3 Logging Content

#### debug\_print\_parse

**Parameter description:** Specifies whether to print parsing tree results.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

**Default value:** off

## debug\_print\_rewritten

**Parameter description:** Specifies whether to print query rewriting results.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

**Default value:** off

## debug\_print\_plan

**Parameter description:** Specifies whether to print the query execution plan to logs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

**Default value:** off

---

### NOTICE

- Debugging information about **debug\_print\_parse**, **debug\_print\_rewritten**, and **debug\_print\_plan** are printed only when the log level is set to **log** or higher. When these parameters are set to **on**, their debugging information will be recorded in server logs and will not be sent to client logs. You can change the log level by setting [client\\_min\\_messages](#) and [log\\_min\\_messages](#).
  - Do not invoke the **gs\_encrypt\_aes128** and **gs\_decrypt\_aes128** functions when **debug\_print\_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the **gs\_encrypt\_aes128** and **gs\_decrypt\_aes128** functions in the log files generated when **debug\_print\_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.
- 

## debug\_pretty\_print

**Parameter description:** Indents the logs produced by **debug\_print\_parse**, **debug\_print\_rewritten**, and **debug\_print\_plan**. The output format is more readable but much longer than that generated when this parameter is set to **off**.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the indentation is enabled.
- **off** indicates that the indentation is disabled.

**Default value:** on

## log\_checkpoints

**Parameter description:** Specifies whether the statistics on checkpoints and restart points are recorded in the server logs. When this parameter is set to **on**, statistics on checkpoints and restart points are recorded in the log messages, including the number of buffers written and the time spent in writing them.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the statistics on checkpoints and restart points are recorded in the server logs.
- **off** indicates that the statistics on checkpoints and restart points are not recorded in the server logs

**Default value:** off

## log\_connections

**Parameter description:** Specifies whether to record connection request information of the client.

This parameter is a BACKEND parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

Some client programs, such as gsql, attempt to connect twice while determining if a password is required. In this case, duplicate "connection receive" messages do not necessarily indicate a problem.

---

**Value range:** Boolean

- **on** indicates that the request information is recorded.
- **off** indicates that the request information is not recorded.

**Default value:** off

## log\_disconnections

**Parameter description:** Specifies whether to record disconnection request information of the client.

This parameter is a BACKEND parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the request information is recorded.
- **off** indicates that the request information is not recorded.

**Default value:** off

## log\_duration

**Parameter description:** Specifies whether to record the duration of every completed SQL statement. For clients using extended query protocols, the time required for parsing, binding, and executing steps are logged independently.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **off:** Compared with this option, [log\\_min\\_duration\\_statement](#) forcibly records the query text.
- If this parameter is set to **on** and [log\\_min\\_duration\\_statement](#) is set to a positive value, the duration of each completed statement is logged but the query text is included only for statements exceeding the threshold. This behavior can be used for gathering statistics in high-load situation.

**Default value:** off

## log\_error\_verbosity

**Parameter description:** Specifies the amount of detail written in the server log for each message that is logged.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **terse** indicates that the output excludes the DETAIL, HINT, QUERY, and CONTEXT error information.
- **verbose** indicates that the output includes the SQLSTATE error code, the source code file name, function name, and number of the line in which the error occurs.
- **default** indicates that the output includes the DETAIL, HINT, QUERY, and CONTEXT error information, and excludes the SQLSTATE error code, the source code file name, function name, and number of the line in which the error occurs.

**Default value:** default

## log\_hostname

**Parameter description:** By default, connection log messages only show the IP address of the connecting host. The host name can be recorded when this parameter is set to **on**. It may take some time to parse the host name. Therefore, the database performance may be affected.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the host name is simultaneously recorded.
- **off** indicates that the host name is not simultaneously recorded.

**Default value:** off

## log\_line\_prefix

**Parameter description:** Specifies the prefix format of each log information. A prefix is a printf-style string that is output at the beginning of each line of the log. The "escape sequences" which begin with % are replaced with status information as listed in [Table 14-7](#).

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Table 14-7** Escape characters

Escape Character	Effect
%a	Application name
%u	Username
%d	Database name
%r	Remote host name or IP address and remote port. If <b>log_hostname</b> is set to <b>off</b> , only the IP address and remote port are displayed.
%h	Remote host name or IP address. If <b>log_hostname</b> is set to <b>off</b> , only the IP address is displayed.
%p	Thread ID
%t	Timestamp without milliseconds (no time zone in the Windows OS)
%m	Timestamp with milliseconds
%n	Node from which an error is reported
%i	Command tag: type of command executed in the current session
%e	SQLSTATE error code
%c	Session ID: For details, see the note below the table.
%l	Number of the log line for each session, starting from 1
%s	Start time of a session
%v	Virtual transaction ID (backendID/ localXID)
%x	Transaction ID ( <b>0</b> indicates that no transaction ID is assigned)

Escape Character	Effect
%q	Produces no output. If the current thread is a backend thread, this escape sequence is ignored and subsequent escape sequences are processed. Otherwise, this escape sequence and subsequent escape sequences are all ignored.
%S	Session ID
%T	Trace ID
%%	The character %

 **NOTE**

The %c escape character prints a unique session ID consisting of two 4-byte hexadecimal numbers separated by a period (.). The numbers are the process startup time and the process ID. Therefore, %c can also be used as a space saving way of printing those items. For example, run the following query to generate the session ID from **pg\_stat\_activity**:

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
 to_hex(pid)
FROM pg_stat_activity;
```

- If you set a non-empty value for **log\_line\_prefix**, ensure that its last character is a space, to provide visual separation from the rest of the log line. A punctuation character can be used, too.
- Syslog generates its own timestamp and process ID information. Therefore, you do not need to include those escapes characters when you are logging in to syslog.

**Value range:** a string

**Default value:** '%m %n %u %d %h %p %S %x %a '

 **NOTE**

**%m %n %u %d %h %p %S %x %a** indicates the session start timestamp, error reporting node, username, database name, remote host name or IP address, thread ID, session ID, transaction ID, and application name.

## log\_lock\_waits

**Parameter description:** If the time for which a session waits to acquire a lock is longer than the value of **deadlock\_timeout**, this parameter specifies whether to record this message in the database. This is useful in determining if lock waits are causing poor performance.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the information is recorded.
- **off** indicates that the information is not recorded.

**Default value:** **off**

## log\_statement

**Parameter description:** Specifies which SQL statements are recorded. For clients using extended query protocols, logging occurs when an Execute message is received, and values of the Bind parameters are included (with any embedded single quotation marks doubled).

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

### NOTICE

- Statements that contain simple syntax errors are not logged even if **log\_statement** is set to **all**, because the log message is emitted only after basic parsing has been completed to determine the statement type. If an extended query protocol is used, statements that fail before the execution phase (during parse analysis or planning) are not logged, either. Set **log\_min\_error\_statement** to **ERROR** or lower to log such statements.
- If this parameter is set to a value other than **none**, the statement audit function is enabled. The database administrator can access server logs to view SQL execution records.

**Value range:** enumerated values

- **none** indicates that no statement is recorded.
- **ddl** indicates that all data definition statements, such as CREATE, ALTER, and DROP, are recorded.
- **mod** indicates that all DDL statements and data modification statements, such as INSERT, UPDATE, DELETE, TRUNCATE, and COPY FROM, are recorded.
- **all** indicates that all statements, including the PREPARE, EXECUTE, and EXPLAIN ANALYZE statements, are recorded.

**Default value:** none

## log\_temp\_files

**Parameter description:** Specifies whether to record the deletion information of temporary files. Temporary files can be created for sorting, hashing, and storing temporary querying results. If the recording is enabled, a log entry is generated for each temporary file when it is deleted.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 2147483647. The unit is KB.

- A positive value indicates that the deletion information of temporary files whose size is larger than the specified value of **log\_temp\_files** is recorded.
- **0** indicates that the delete information of all temporary files is recorded.
- **-1** indicates that the delete information of any temporary files is not recorded.

**Default value:** -1

## log\_timezone

**Parameter description:** Specifies the time zone used for timestamps written in the server log. Different from **TimeZone**, this parameter takes effect for all sessions in the database.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

**Value range:** a string. You can query the PG\_TIMEZONE\_NAMES view to obtain the value. For details, see "System Catalogs and System Views > System Views > PG\_TIMEZONE\_NAMES" in *Developer Guide*.

**Default value:** Set this parameter based on the OS time zone.

### NOTE

The default value will be changed when **gs\_initdb** is used to set system environments.

## logging\_module

**Parameter description:** Specifies whether module logs are output on the server. This parameter is a session-level parameter, and you are advised not to use the **gs\_guc** tool to set it.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

**Value range:** a string

**Default value:** Logs of the LOCK module are generated on the server. Logs of other modules are not generated on the server, but can be viewed by running **SHOW logging\_module**.

```
ALL,on(LOCK),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GD
S,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRIT
E,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UD
F,COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,H
ANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPS
HOT,WDR_REPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PAR
AM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTO
RE,UPAGE,UBTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGE
R,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWN
ER,GSSSTACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT
NEWPAGE,GPI,GS_DEPENDENCY,LWLOCK,GS)
```

---

### CAUTION

**CN\_RETRY** does not take effect in the current version.

---

**Setting method:** Run **SHOW logging\_module** to view which modules are controllable. For example, the query output result is as follows:

```
gaussdb=# show logging_module;
logging_module
```

```

ALL,on(LOCK),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GD
S,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRIT
```

```
E,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UD
F,COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,H
ANDLE,CLOG,EC,REMOTE,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_R
EPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESER
IES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,U
BTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLIC
Y,SEC_SDD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSSTACK,
LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPAGE,G
PI,GS_DEPENDENCY,LWLOCK,GS)
```

(1 row)

Controllable modules are identified by uppercase letters, and the special ID **ALL** is used for setting all module logs. You can control the output of module logs by setting **logging\_module** to **on** or **off**. Enable log output for SSL:

```
gaussdb=# set logging_module='on(SSL)';
SET
gaussdb=# show
logging_module;
 logging_module

ALL,on(SSL,LOCK),off(COMMAND,DFS,GUC,GS CLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,GD
S,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRIT
E,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UD
F,COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,H
ANDLE,CLOG,EC,REMOTE,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_R
EPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESER
IES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,U
BTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLIC
Y,SEC_SDD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSSTACK,
LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPAGE,G
PI,GS_DEPENDENCY,LWLOCK,GS)
```

(1 row)

SSL log output is enabled.

The **ALL** identifier can be used to quickly enable or disable log output for all modules.

```
gaussdb=# set logging_module='off(ALL)';
SET
gaussdb=# show
logging_module;
 logging_module

ALL,on(),off(COMMAND,DFS,GUC,GS CLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBL
SPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT
_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UDF,CO
OP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDL
E,CLOG,EC,REMOTE,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_REPOR
T,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SC
HEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,
UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_S
DD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSSTACK,LOGICA
L_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPAGE,GPI,GS_D
EPENDENCY,LWLOCK,LOCK,GS)
```

(1 row)

```
gaussdb=# set logging_module='on(ALL)';
SET
gaussdb=# show
logging_module;
 logging_module

```

```

ALL,on(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,
WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOI
N,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UDF,COOP_A
NALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDLE,CL
OG,EC,REMOTE,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_REPORT,IN
CRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHE
MA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,U
NDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_S
D,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSSTACK,LOGICAL
_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPAGE,GPI,GS_DE
PENDENCY,LWLOCK,LOCK,GS),off()
(1 row)
```

**Dependency:** The value of this parameter depends on the settings of **log\_min\_level**.

## enable\_unshipping\_log

**Parameter description:** Specifies whether to log statements that are not pushed down. The logs help locate performance issues that may be caused by statements not pushed down. If **enable\_stream\_operator** is set to **off** and this parameter is set to **on**, a large number of logs indicating that plans cannot be pushed down are recorded. If you do not need these logs, you are advised to set both **enable\_unshipping\_log** and **enable\_stream\_operator** to **off**.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that statements not pushed down are logged.
- **off** indicates that statements not pushed down are not logged.

**Default value:** **off**

## opfusion\_debug\_mode

**Parameter description:** Checks whether simple queries are optimized for debugging. If this parameter is set to **log**, you can view the specific reasons why queries are not optimized in the DN execution plans.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **off** indicates that reasons why queries are not optimized are not included.
- **log** indicates that reasons why queries are not optimized are included in the DN execution plan.

#### NOTICE

- You need to set **max\_datanode\_for\_plan** to view the DN execution plans.
- To view the reasons why queries are not optimized in the log, set **opfusion\_debug\_mode** to **log**, **log\_min\_messages** to **debug4**, and **logging\_module** to **on(OPFUSION)**. Note that a large number of log messages may be generated. Therefore, execute only a small number of jobs during debugging.

**Default value:** off

### enable\_debug\_vacuum

**Parameter description:** Specifies whether to allow output of some VACUUM-related logs for problem locating. This parameter is used only by developers. Common users are advised not to use it.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on/true** indicates that output of VACUUM-related logs is allowed.
- **off/false** indicates that output of VACUUM-related logs is disallowed.

**Default value:** off

### resource\_track\_log

**Parameter description:** Specifies the log level of self-diagnosis. Currently, this parameter takes effect only in multi-column statistics.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

- **summary:** Brief diagnosis information is displayed.
- **detail:** Detailed diagnosis information is displayed.

Currently, the two parameter values differ only when there is an alarm about multi-column statistics not collected. If the parameter is set to **summary**, such an alarm will not be displayed. If it is set to **detail**, such an alarm will be displayed.

**Default value:** summary

#### 14.3.9.4 Using CSV Log Output

##### Prerequisites

- The **log\_destination** parameter is set to **csvlog**.
- The **logging\_collector** parameter is set to **on**.

## Definition of csvlog

Log lines are emitted in comma-separated values (CSV) format.

An example table definition for storing CSV-format log output is shown as follows:

```
CREATE TABLE postgres_log
(
log_time timestamp(3) with time zone,
node_name text,
user_name text,
database_name text,
process_id bigint,
connection_from text,
"session_id" text,
session_line_num bigint,
command_tag text,
session_start_time timestamp with time zone,
virtual_transaction_id text,
transaction_id bigint,
query_id bigint,
module text,
error_severity text,
sql_state_code text,
message text,
detail text,
hint text,
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text
);
```

For details, see [Table 14-8](#).

**Table 14-8** Meaning of each csvlog field

Field	Description	Field	Description
log_time	Timestamp in milliseconds	module	Log module
node_name	Node name	error_severity	ERRORSTATE code
user_name	Username	sql_state_code	SQLSTATE code
database_name	Database name	message	Error message
process_id	Process ID	detail	Detailed error message
connection_from	Port number of the client host	hint	Prompt message

Field	Description	Field	Description
session_id	Session ID	internal_query	Internal query (This field is used to query the information leading to errors if any.)
session_line_num	Number of lines in each session	internal_query_pos	Pointer for an internal query
command_tag	Command tag	context	Environment
session_start_time	Start time of a session	query	Character count at the position where errors occur
virtual_transaction_id	Regular transaction	query_pos	Pointer at the position where errors occur
transaction_id	Transaction ID	location	Position where errors occur in the GaussDB source code if <b>log_error_verbosity</b> is set to <b>verbose</b>
query_id	Query ID	application_name	Application name

Run the following command to import a log file to this table:  
`COPY postgres_log FROM '/opt/data/pg_log/logfile.csv' WITH csv;`

 **NOTE**

The log name (**logfile.csv**) here needs to be replaced with the name of a log generated.

## Simplifying Input

Simplify importing CSV log files by performing the following operations:

- Set **log\_filename** and **log\_rotation\_age** to provide a consistent, predictable naming solution for log files. By doing this, you can predict when an individual log file is complete and ready to be imported.
- Set **log\_rotation\_size** to **0** to disable size-based log rollback, as it makes the log file name difficult to predict.
- Set **log\_truncate\_on\_rotation** to **on** so that old log data cannot be mixed with the new one in the same file.

### 14.3.10 Alarm Detection

During the running of the cluster, error scenarios can be detected and informed to users in a timely manner. You can view the **system\_alarm** log written by the alarm in the `$GAUSSLOG/cm` or the `$GAUSSLOG/pg_log/gtm` directory.

## enable\_alarm

**Parameter description:** Specifies whether to enable the alarm detection thread to detect fault scenarios that may occur in the database.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the alarm detection thread is enabled.
- **off** indicates that the alarm detection thread is disabled.

**Default value:** on

### NOTE

This parameter takes effect only on CNs and DNPs.

## connection\_alarm\_rate

**Parameter description:** Specifies the ratio restriction on the maximum number of allowed parallel connections to the database. The maximum number of concurrent connections to the database is **max\_connections** x **connection\_alarm\_rate**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from 0.0 to 1.0

**Default value:** 0.9

## alarm\_report\_interval

**Parameter description:** specifies the interval at which an alarm is reported.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer. The unit is s.

**Default value:** 10

## alarm\_component

**Parameter description:** Certain alarms are suppressed during alarm reporting. That is, the same alarm will not be repeatedly reported by an instance within the period specified by **alarm\_report\_interval**. Its default value is **10s**. In this case, the parameter specifies the location of the alarm component that is used to process alarm information. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

- If **--alarm-type** in the **gs\_preinstall** script is set to **5**, no third-party component is connected and alarms are written into the **system\_alarm** log.

In this case, the value of **alarm\_component** is `/opt/huawei/snas/bin/snas_cm_cmd`.

- If `--alarm-type` in the `gs_preinstall` script is set to `1`, a third-party component is connected. In this case, the value of **alarm\_component** is the absolute path of the executable program of the third-party component.

**Default value:** `/opt/huawei/snas/bin/snas_cm_cmd`

## 14.3.11 Statistics During the Database Running

### 14.3.11.1 Query and Index Statistics Collector

The query and index statistics collector is used to collect statistics during database running. The statistics include the times of inserting and updating a table and index, the number of disk blocks and tuples, and the time required for the last cleanup and analysis on each table. The statistics can be viewed by querying system view `pg_stats` and `pg_statistic`. The following parameters are used to set the statistics collection feature in the server scope.

#### **track\_activities**

**Parameter description:** Collects statistics about the commands that are being executed in session. For a stored procedure, if this parameter is enabled, you can view the PERFORM statement, stored procedure calling statement, SQL statement, and OPEN CURSOR statement that are being executed in the stored procedure in the `pg_stat_activity` view.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

**Default value:** **on**

#### **track\_counts**

**Parameter description:** Collects statistics about database activities.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

#### NOTE

When the database to be cleaned up is selected from the **AutoVacuum** automatic cleanup process, the database statistics are required. In this case, the default value is set to **on**.

**Default value:** **on**

## track\_io\_timing

**Parameter description:** Collects statistics about I/O timing in the database. The I/O timing statistics can be queried by using the **pg\_stat\_database** parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- If this parameter is set to **on**, the collection function is enabled. In this case, the collector repeatedly queries the operating system at the current time. As a result, large number of costs may occur on some platforms. Therefore, the default value is set to **off**.
- **off** indicates that the statistics collection function is disabled.

**Default value:** off

## track\_functions

**Parameter description:** Collects statistics of the number and duration of function invocations.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

When the SQL functions are set to inline functions queried by the invoking, these SQL functions cannot be traced no matter these functions are set or not.

---

**Value range:** enumerated values

- **pl** indicates that only procedural language functions are traced.
- **none** indicates that the function tracing function is disabled.

**Default value:** none

## track\_activity\_query\_size

**Parameter description:** Specifies byte counts of the current running commands used to trace each active session.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 100 to 102400, in bytes

**Default value:** 1024

## update\_process\_title

**Parameter description:** Collects statistics updated with a process name each time the server receives a new SQL statement.

The process name can be viewed on Windows task manager by running the **ps** command.

This parameter is an INTERNAL parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

**Default value:** off

## stats\_temp\_directory

**Parameter description:** Specifies the directory for storing temporary statistics. Only the **sysadmin** user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

If a RAM-based file system directory is used, the actual I/O cost can be lowered and the performance can be improved.

---

**Value range:** a string

**Default value:** pg\_stat\_tmp

## track\_thread\_wait\_status\_interval

**Parameter description:** Specifies the interval of collecting the thread status information.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1440. The unit is min.

**Default value:** 30min

## enable\_save\_datachanged\_timestamp

**Parameter description:** Specifies whether to record the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE/TRUNCATE/DROP PARTITION** is performed on table data.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the time when an operation is performed on table data will be recorded.

- **off** indicates that the time when an operation is performed on table data will not be recorded.

**Default value:** on

## plan\_collect\_thresh

**Parameter description:** Collects statistics about the plans that are being executed in each session.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 2147483647

- **-1:** Plans that are being executed are not collected.
- **0:** Plans that are being executed are collected before plan execution.
- A value greater than **0** indicates that when the total number of tuples incrementally returned by all operators in a plan is greater than or equal to the value of this parameter, plans that are being executed are collected once.

**Default value:** 0

## track\_sql\_count

**Parameter description:** Collects statistics on the statements (**SELECT**, **INSERT**, **UPDATE**, **MERGE INTO**, and **DELETE**) that are being executed in a session.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 0.8% by enabling or disabling this parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the auditing function is disabled.

**Default value:** on

### NOTE

If **track\_sql\_count** is set to **off**, querying the **gs\_sql\_count** or **pgxc\_sql\_count** view returns 0.

### 14.3.11.2 Performance Statistics

During the running of the database, the lock access, disk I/O operation, and invalid message processing are involved. All these operations are the bottleneck of the database performance. The performance statistics provided by GaussDB can facilitate the performance fault location.

## Generating Performance Statistics Logs

**Parameter description:** For each query, the following four parameters record the performance statistics of corresponding modules in the server log:

- The **log\_parser\_stats** parameter records the performance statistics of a parser in the server log.
- The **log\_planner\_stats** parameter records the performance statistics of a query optimizer in the server log.
- The **log\_executor\_stats** parameter records the performance statistics of an executor in the server log.
- The **log\_statement\_stats** parameter records the performance statistics of the whole statement in the server log.

All these parameters can only provide assistant analysis for administrators, which are similar to the `getrusage()` of the Linux OS.

These parameters are SUSET parameters. Set them based on instructions provided in [Table 14-1](#).

---

### NOTICE

- The **log\_statement\_stats** records the total statement statistics whereas other parameters record statistics only about their corresponding modules.
- The **log\_statement\_stats** parameter cannot be enabled together with any parameter recording statistics about a module.

---

**Value range:** Boolean

- **on** indicates that performance statistics are recorded.
- **off** indicates that performance statistics are not recorded.

**Default value:** off

### 14.3.11.3 Hotspot Key Statistics

In the distributed architecture, if applications access a node in a short period of time, the resource usage of the node is too high, affecting the normal running of the database. GaussDB provides the function of quickly detecting hotspot keys to quickly determine whether there are hotspot keys and the distribution of hotspot keys.

#### enable\_hotkeys\_collection

**Parameter description:** Specifies whether to automatically collect statistics on the accessed key values in the database.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

 NOTE

If you set parameters using **gs\_guc** set, you need to restart the database for the GUC parameters to take effect. During the restart, hotspot key information is cleared.

When the GUC parameter is disabled, the query result of the hotspot key is empty and a message is displayed indicating that the GUC parameter is disabled. However, when the function is disabled, the hotspot key clearance API can still be used.

**Value range:** Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the auditing function is disabled.

**Default value:** off

## 14.3.12 Automatic Vacuuming

The **autovacuum** process automatically runs the **VACUUM** and **ANALYZE** statements to recycle the record space marked as deleted and update statistics about the table.

### autovacuum

**Parameter description:** Specifies whether to start the **autovacuum** process in the database. Ensure that the **track\_counts** parameter is set to **on** before starting the **autovacuum** process.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

 NOTE

- Set the **autovacuum** parameter to **on** to automatically vacuum two-phase transactions after the system recovers from faults.
- If **autovacuum** is set to **on** and **autovacuum\_max\_workers** to **0**, the autovacuum process is enabled only when the system recovers from faults to clean up abnormal two-phase transactions.
- If **autovacuum** is set to **on** and **autovacuum\_max\_workers** to a value greater than **0**, the **autovacuum** process is started to clean up two-phase transactions and processes when the system recovers from faults.

**Value range:** Boolean

- **on** indicates that the **autovacuum** process is started.
- **off** indicates that the **autovacuum** process is not started.

**Default value:** on

### autovacuum\_mode

**Parameter description:** Specifies whether the autoanalyze or autovacuum function is enabled. This parameter is valid only when **autovacuum** is set to **on**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **analyze** indicates that only autoanalyze is performed.
- **vacuum** indicates that only autovacuum is performed.
- **mix** indicates that both autoanalyze and autovacuum are performed.
- **none** indicates that neither of them is performed.

**Default value:** mix

## autoanalyze\_timeout

**Parameter description:** Specifies the timeout period of autoanalyze. If the duration of autoanalyze on a table exceeds the value of **autoanalyze\_timeout**, the autoanalyze operation is automatically canceled.

The timeout check cannot be completely accurate. In principle, the statistics on each CN must be consistent. Therefore, the synchronization between CNs will not be interrupted even if the synchronization times out. As a result, the actual execution time may exceed the user-defined time.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483, in seconds. The value 0 indicates no timeout.

**Default value:** 5min (300s)

## autovacuum\_io\_limits

**Parameter description:** Specifies the upper limit of I/Os triggered by the autovacuum process per second.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 1073741823. -1 indicates that the default Cgroup is used.

**Default value:** -1

## log\_autovacuum\_min\_duration

**Parameter description:** Records each step performed by the autovacuum process to the server log when the execution time of the autovacuum process is greater than or equal to a certain value. This parameter helps track the autovacuum behavior.

For example, set the **log\_autovacuum\_min\_duration** parameter to **250ms** to record the information about the autovacuum commands running longer than or equal to 250 ms.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 2147483647. The unit is ms.

- 0 indicates that all autovacuum actions are recorded in the log.

- **-1** indicates that all autovacuum actions are not recorded in the log.
- A value other than **-1** indicates that a message is recorded when an autovacuum action is skipped due to a lock conflict.

**Default value:** -1

## autovacuum\_max\_workers

**Parameter description:** Specifies the maximum number of autovacuum worker threads that can run at the same time. The upper limit of this parameter is related to the values of **max\_connections** and **job\_queue\_processes**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer. The minimum value is **0**, indicating that autovacuum is not enabled. The theoretical maximum value is **262143**, but the actual maximum value is a dynamic value calculated by the following formula:  $262143 - \text{Value of } \mathbf{max\_inner\_tool\_connections} - \text{Value of } \mathbf{max\_connections} - \text{Value of } \mathbf{max\_concurrent\_autonomous\_transactions} - \text{Value of } \mathbf{job\_queue\_processes} - \text{Number of auxiliary threads} - \text{Number of autovacuum launcher threads} - 1$ . The number of auxiliary threads and the number of autovacuum launcher threads are specified by two macros. Their default values are **20** and **2** respectively.

**Default value:** 3

## autovacuum\_naptime

**Parameter description:** Specifies the interval between activity rounds for the autovacuum process.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483. The unit is s.

**Default value:** 10min (600s)

## autovacuum\_vacuum\_threshold

**Parameter description:** Specifies the threshold for triggering the VACUUM operation. When the number of deleted or updated records in a table exceeds the specified threshold, the VACUUM operation is executed on this table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647.

**Default value:** 50

## autovacuum\_analyze\_threshold

**Parameter description:** Specifies the threshold for triggering the ANALYZE operation. When the number of deleted, inserted, or updated records in a table exceeds the specified threshold, the ANALYZE operation is executed on this table.

If a global secondary index exists in a table, the ANALYZE operation is performed on the global secondary index only when the number of deleted, inserted, and updated records in the table exceeds the threshold 20 times.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647.

**Default value:** 50

## autovacuum\_vacuum\_scale\_factor

**Parameter description:** Specifies a fraction of the table size added to the **autovacuum\_vacuum\_threshold** parameter when deciding whether to vacuum a table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating-point number ranging from 0.0 to 100.0

**Default value:** 0.2

## autovacuum\_analyze\_scale\_factor

**Parameter description:** Specifies a fraction of the table size added to the **autovacuum\_analyze\_threshold** parameter when deciding whether to analyze a table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating-point number ranging from 0.0 to 100.0

**Default value:** 0.1

## autovacuum\_freeze\_max\_age

**Parameter description:** Specifies the maximum age (in transactions) that a table's **pg\_class.relfrozexid** column can attain before a VACUUM operation is performed.

- The old files under the subdirectory of **pg\_clog/** can also be deleted by the VACUUM operation.
- Even if the autovacuum process is not started, the system will invoke the process to prevent transaction ID wraparound.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 100000 to 576460752303423487

**Default value:** 4000000000

## autovacuum\_vacuum\_cost\_delay

**Parameter description:** Specifies the value of the cost delay used in the autovacuum operation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 100. The unit is ms. -1 indicates that the regular vacuum cost delay is used.

**Default value:** 20 ms

## autovacuum\_vacuum\_cost\_limit

**Parameter description:** Sets the value of the cost limit used in the autovacuum operation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 10000 -1 indicates that the regular vacuum cost limit is used.

**Default value:** -1

## twophase\_clean\_workers

**Parameter description:** Specifies the maximum number of concurrent cleanup operations that can be performed by the gs\_clean tool.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 10

**Default value:** 3

## defer\_csn\_cleanup\_time

**Parameter description:** Specifies the local recycling interval.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is ms.

**Default value:** 5000 ms (that is, 5s)

## 14.3.13 Default Settings of Client Connection

### 14.3.13.1 Statement Behavior

This section describes related default parameters involved in the execution of SQL statements.

## search\_path

**Parameter description:** Specifies the order in which schemas are searched when an object is referenced with no schema specified. The value of this parameter consists of one or more schema names. Different schema names are separated by commas (,).

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

- If the schema of temporary tables exists in the current session, the scheme can be listed in **search\_path** by using the alias **pg\_temp**, for example, '**pg\_temp,public**'. The schema of temporary tables has the highest search priority and is always searched before all the other schemas specified in **pg\_catalog** and **search\_path**. Therefore, do not explicitly specify **pg\_temp** to be searched after other schemas in **search\_path**. This setting will not take effect and an error message will be displayed. If the alias **pg\_temp** is used, the temporary schema will be searched only for tables, views, and data types, and not for functions or operators.
- The system catalog schema, **pg\_catalog**, has the second highest search priority and is the first to be searched among all the schemas, excluding **pg\_temp**, specified in **search\_path**. Therefore, do not explicitly specify **pg\_catalog** to be searched after other schemas in **search\_path**. This setting will not take effect and an error message will be displayed.
- When an object is created without a specific target schema, the object will be placed in the first valid schema listed in **search\_path**. An error is reported if the search path is empty.
- The current effective value of the search path can be examined through the SQL function **current\_schema**. This is different from examining the value of **search\_path**, because the **current\_schema** function displays the first valid schema name in **search\_path**.

**Value range:** a string

### NOTE

- When this parameter is set to "**\$user**", **public**, shared use of a database (where no users have private schemas, and all share use of public), private per-user schemas and combinations of them are supported. Other effects can be obtained by modifying the default search path setting, either globally or per-user.
- When this parameter is set to a null string (""), the system automatically converts it into a pair of double quotation marks ("").
- If the content contains double quotation marks, the system considers them as insecure characters and converts each double quotation mark into a pair of double quotation marks.

**Default value:** "**\$user**",**public**

### NOTE

**\$user** indicates the name of the schema with the same name as the current session user. If the schema does not exist, **\$user** will be ignored.

## current\_schema

**Parameter description:** Specifies the current schema.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** "\$user",public

 NOTE

**\$user** indicates the name of the schema with the same name as the current session user. If the schema does not exist, **\$user** will be ignored.

## default\_tablespace

**Parameter description:** Specifies the default tablespace of the created objects (tables and indexes) when a **CREATE** command does not explicitly specify a tablespace.

- The value of this parameter is either the name of a tablespace, or an empty string that indicates the use of the default tablespace of the current database. If a non-default tablespace is specified, users must have CREATE privilege for it. Otherwise, creation attempts will fail.
- This parameter is not used for temporary tables. For them, [temp\\_tablespaces](#) is used instead.
- This parameter is not used when users create databases. By default, a new database inherits its tablespace setting from the template database.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. An empty string indicates that the default tablespace is used.

**Default value:** empty

## default\_storage\_nodegroup

**Parameter description:** Specifies the Node Group where a table is created by default. This parameter takes effect only for ordinary tables.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

- **installation** indicates that tables will be created in the Node Group created during database installation.
- A value other than **installation** indicates that tables will be created in the Node Group specified by this parameter.

**Value range:** a string

**Default value:** installation

## temp\_tablespaces

**Parameter description:** Specifies one or more tablespaces to which temporary objects (temporary tables and their indexes) will be created when a CREATE

command does not explicitly specify a tablespace. Temporary files for sorting large data sets are created in these tablespaces.

The value of this parameter can be a list of names of tablespaces. When there is more than one name in the list, GaussDB chooses a random tablespace from the list upon the creation of a temporary object each time. However, within a transaction, successively created temporary objects are placed in successive tablespaces in the list. If the element selected from the list is an empty string, GaussDB will automatically use the default tablespace of the current database instead.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string An empty string indicates that all temporary objects are created only in the default tablespace of the current database. For details, see [default\\_tablespace](#).

**Default value:** empty

## check\_function\_bodies

**Parameter description:** Specifies whether to enable validation of the function body string during the execution of **CREATE FUNCTION**. Verification is occasionally disabled to avoid problems, such as forward references when you restore function definitions from a dump. After the function is enabled, the word syntax of the PL/SQL in the stored procedure is verified, including the data type, statement, and expression. The SQL statements in the stored procedure are not checked in the Create phase. Instead, they are checked during running.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that validation of the function body string is enabled during the execution of **CREATE FUNCTION**.
- **off** indicates that validation of the function body string is disabled during the execution of **CREATE FUNCTION**.

**Default value:** on

## default\_transaction\_isolation

**Parameter description:** Specifies the default isolation level of each transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

### NOTE

The current version does not support the setting of the default transaction isolation level. The default value is **read committed**. Do not change the value.

**Value range:** enumerated values

- **read uncommitted** indicates that a transaction reads the uncommitted modifications made by other transactions.

- **read committed** indicates that the data read by a transaction is committed at the moment it is read.
- **repeatable read** indicates that the data that has been read by the current transaction cannot be modified by other transactions until the current transaction completes, thereby preventing unrepeatable reads.
- **serializable**: Currently, this isolation level is not supported in GaussDB. It is equivalent to **repeatable read**.

**Default value:** read committed

## default\_transaction\_read\_only

**Parameter description:** Specifies whether each new transaction is in read-only state.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).



If this parameter is set to **on**, the DML and write transactions cannot be executed.

---

**Value range:** Boolean

- **on** indicates that the transaction is in read-only state.
- **off** indicates that the transaction is in read/write state.

**Default value:** off

## default\_transaction\_deferrable

**Parameter description:** Specifies the default deferrable status of each new transaction. It currently has no effect on read-only transactions or those running at isolation levels lower than serializable.

GaussDB does not support the serializable isolation level. Therefore, the parameter takes no effect.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that a transaction is delayed by default.
- **off** indicates that a transaction is not delayed by default.

**Default value:** off

## session\_replication\_role

**Parameter description:** Specifies the behavior of replication-related triggers and rules for the current session.

This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

Setting this parameter will discard all the cached execution plans.

---

**Value range:** enumerated values

- **origin** indicates that the system copies operations such as insert, delete, and update from the current session.
- **replica** indicates that the system copies operations such as insert, delete, and update from other places to the current session.
- **local** indicates that the system will detect the role that has logged in to the database when using the function to copy operations and will perform related operations.

**Default value:** origin

## statement\_timeout

**Parameter description:** If the statement execution time (starting from the time the server receives the command) is longer than the duration specified by the parameter, error information is displayed and the statement exits.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#). The default value is 0, indicating that the parameter does not take effect.

**Value range:** an integer ranging from 0 to 2147483647. The unit is ms.

**Default value:** 0

## vacuum\_freeze\_min\_age

**Parameter description:** Specifies whether VACUUM replaces the xmin column of a record with FrozenXID when scanning a table (in the same transaction).

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 576460752303423487

 **NOTE**

Although you can set this parameter to any value, **VACUUM** will limit the effective value to half the value of [autovacuum\\_freeze\\_max\\_age](#) by default.

**Default value:** 200000000

## vacuum\_freeze\_table\_age

**Parameter description:** Specifies when VACUUM scans the whole table and freezes old tuples. VACUUM performs a full table scan if the difference between the current transaction ID and the value of pg\_class.relfrozexid64 is greater than the specified time.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 576460752303423487

 **NOTE**

Although you can set this parameter to any value, **VACUUM** will limit the effective value to 95% of **autovacuum\_freeze\_max\_age** by default. Therefore, a periodic manual VACUUM has a chance to run before an anti-wraparound autovacuum is launched for the table.

**Default value:** 4000000000

## bytea\_output

**Parameter description:** Specifies the output format for values of the bytea type.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **hex** indicates that the binary data is converted to hexadecimal format.
- **escape** indicates that the traditional PostgreSQL format is used. It takes the approach of representing a binary string as a sequence of ASCII characters, while converting those bytes that cannot be represented as an ASCII character into special escape sequences.

**Default value:** hex

## xmlbinary

**Parameter description:** Specifies how binary values are to be encoded in XML.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

 **NOTE**

Currently, this parameter does not support data of the XML type.

**Value range:** enumerated values

- base64
- hex

**Default value:** base64

## xmloption

**Parameter description:** Specifies whether DOCUMENT or CONTENT is implicit when converting between XML and string values.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

 **NOTE**

Currently, this parameter does not support data of the XML type.

**Value range:** enumerated values

- **document** indicates an HTML document.
- **content** indicates a common string.

**Default value:** content

### max\_compile\_functions

**Parameter description:** Specifies the maximum number of function compilation results stored in the server. Excessive functions and compilation results of stored procedures may occupy large memory space. Setting this parameter to a proper value can reduce the memory usage and improve system performance.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647

**Default value:** 1000

### gin\_pending\_list\_limit

**Parameter description:** Specifies the maximum size of the GIN pending list which is used when **fastupdate** is enabled. If the list grows larger than this maximum size, it is cleaned up by moving the entries in it to the main GIN data structure in batches. This setting can be overridden for individual GIN indexes by changing index storage parameters.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 64 to 2147483647. The unit is KB.

**Default value:** 4 MB

## 14.3.13.2 Locale and Formatting

This section describes parameters related to the time format setting.

### DateStyle

**Parameter description:** Specifies the display format for date and time values, as well as the rules for interpreting ambiguous date input values.

This variable contains two independent components: the output format specifications (ISO, Postgres, SQL, or German) and the input/output order of year/month/day (DMY, MDY, or YMD). The two components can be set separately or together. The keywords Euro and European are synonyms for DMY; the keywords US, NonEuro, and NonEuropean are synonyms for MDY.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** 'ISO, MDY'

 NOTE

**gs\_initdb** will initialize this parameter so that its value is the same as that of **lc\_time**.

If this parameter is modified by running the **gs\_guc reload** command and the connection of a session on the current node is not from the client but from another node in the cluster to which the node belongs, this parameter does not take effect immediately on the session after the **gs\_guc reload** command is executed. The setting takes effect only after the connection node is disconnected and then reconnected.

**Setting Suggestions:** The ISO format is recommended. Postgres, SQL, and German use abbreviations for time zones, such as **EST**, **WST**, and **CST**. These abbreviations can be ambiguous. For example, **CST** can represent Central Standard Time (USA) UT-6:00, Central Standard Time (Australia) UT+9:30, and China Standard Time UT+8:00. This may lead to incorrect time zone conversion and cause errors.

## IntervalStyle

**Parameter description:** Specifies the display format for interval values.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **sql\_standard** indicates that output matching SQL standards will be generated.
- **postgres** indicates that output matching PostgreSQL 8.4 will be generated when the **DateStyle** parameter is set to **ISO**.
- **postgres\_verbose** indicates that output matching PostgreSQL 8.4 will be generated when the **DateStyle** parameter is set to **non\_ISO**.
- **iso\_8601** indicates that output matching the time interval "format with designators" defined in ISO 8601 will be generated.
- **oracle** indicates the output that matches the numtodsinterval function in the Oracle Database. For details, see "SQL Reference > Functions and Operators > Date and Time Processing Functions and Operators > numtodsinterval" in *Developer Guide*.

---

**NOTICE**

The **IntervalStyle** parameter also affects the interpretation of ambiguous interval input.

If this parameter is modified by running the **gs\_guc reload** command and the connection of a session on the current node is not from the client but from another node in the cluster to which the node belongs, this parameter does not take effect immediately on the session after the **gs\_guc reload** command is executed. The setting takes effect only after the connection node is disconnected and then reconnected.

---

**Default value:** postgres

## TimeZone

**Parameter description:** Specifies the time zone for displaying and interpreting timestamps.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. You can query the PG\_TIMEZONE\_NAMES view to obtain the value. For details, see "System Catalogs and System Views > System Views > PG\_TIMEZONE\_NAMES" in *Developer Guide*.

**Default value:**

### NOTE

**gs\_initdb** will set a time zone value that is consistent with the system environment.

If this parameter is modified by running the **gs\_guc reload** command and the connection of a session on the current node is not from the client but from another node in the cluster to which the node belongs, this parameter does not take effect immediately on the session after the **gs\_guc reload** command is executed. The setting takes effect only after the connection node is disconnected and then reconnected.

## timezone\_abbreviations

**Parameter description:** Specifies the time zone abbreviations that will be accepted by the server.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. You can obtain it by querying the **pg\_timezone\_names** view.

**Default value:** Default

### NOTE

**Default** indicates abbreviations that work in most of the world. There are also other abbreviations, such as **Australia** and **India** that can be defined for a particular installation.

## extra\_float\_digits

**Parameter description:** Adjusts the number of digits displayed for floating-point values, including float4, float8, and geometric data types. The parameter value is added to the standard number of digits (FLT\_DIG or DBL\_DIG as appropriate).

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -15 to 3

### NOTE

- This parameter can be set to **3** to include partially-significant digits. It is especially useful for dumping float data that needs to be restored exactly.
- This parameter can also be set to a negative value to suppress unwanted digits.

**Default value:** 0

## client\_encoding

**Parameter description:** Specifies the client-side encoding (character set).

Set this parameter based on the situation of the front-end services. Try to keep the encoding consistent on the client and server to improve efficiency.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** encoding compatible with PostgreSQL. **UTF8** indicates that the database encoding is used.

### NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs\_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.
- To use consistent encoding for communication within the cluster, you are advised to retain the default value of **client\_encoding**. Modification to this parameter in the **postgresql.conf** file (by using the **gs\_guc** tool, for example) does not take effect.

**Default value:** UTF8

**Recommended value:** SQL\_ASCII or UTF8

## lc\_messages

**Parameter description:** Specifies the language in which messages are displayed.

- Acceptable values are system-related.
- On some systems, this locale category does not exist. Setting this variable will still work, but there will be no effect. In addition, translated messages for the desired language may not exist. In this case, you can still see the English messages.

This is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

### NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs\_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

**Default value:** C

## lc\_monetary

**Parameter description:** Specifies the display format of monetary values. It affects the output of functions such as **to\_char**. Acceptable values are system-related.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

 NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs\_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.
- If this parameter is modified by running the **gs\_guc reload** command and the connection of a session on the current node is not from the client but from another node in the cluster to which the node belongs, this parameter does not take effect immediately on the session after the **gs\_guc reload** command is executed. The setting takes effect only after the connection node is disconnected and then reconnected.

**Default value:** C

## lc\_numeric

**Parameter description:** Specifies the display format of numbers. It affects the output of functions such as **to\_char**. Acceptable values are system-related.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

 NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs\_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.
- If this parameter is modified by running the **gs\_guc reload** command and the connection of a session on the current node is not from the client but from another node in the cluster to which the node belongs, this parameter does not take effect immediately on the session after the **gs\_guc reload** command is executed. The setting takes effect only after the connection node is disconnected and then reconnected.

**Default value:** C

## lc\_time

**Parameter description:** Specifies the display format of time and locale. It affects the output of functions such as **to\_char**. Acceptable values are system-related.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

 NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs\_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.
- If this parameter is modified by running the **gs\_guc reload** command and the connection of a session on the current node is not from the client but from another node in the cluster to which the node belongs, this parameter does not take effect immediately on the session after the **gs\_guc reload** command is executed. The setting takes effect only after the connection node is disconnected and then reconnected.

**Default value:** C

## default\_text\_search\_config

**Parameter description:** Specifies the text search configuration.

If the specified text search configuration does not exist, an error will be reported. If the specified text search configuration is deleted, set **default\_text\_search\_config** again. Otherwise, an error will be reported, indicating incorrect configuration.

- The text search configuration is used by text search functions that do not have an explicit argument specifying the configuration.
- When a configuration file matching the environment is determined, **gs\_initdb** will initialize the configuration file with a setting that corresponds to the environment.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

 NOTE

GaussDB supports the following two configurations: **pg\_catalog.english** and **pg\_catalog.simple**.

**Default value:** **pg\_catalog.english**

### 14.3.13.3 Other Default Parameters

This section describes the default database loading parameters.

## dynamic\_library\_path

**Parameter description:** Specifies the path that the system will search for a shared database file that is dynamically loadable. When a dynamically loadable module needs to be opened and the file name specified in the **CREATE FUNCTION** or **LOAD** command does not have a directory component, the system will search this path for the required file. Only the sysadmin user can access this parameter.

The value of **dynamic\_library\_path** must be a list of absolute paths separated by colons (:) or by semi-colons (;) on the Windows OS. When the name of a path starts with the special variable \$libdir, the variable will be replaced with the directory in which the module provided by the GaussDB is installed. For example:

```
dynamic_library_path = '/usr/local/lib/postgresql/opt/testgs/lib:$libdir'
```

This is a SUSE parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

 **NOTE**

If the value of this parameter is set to an empty character string, the automatic path search is turned off.

**Default value:** \$libdir

## gin\_fuzzy\_search\_limit

**Parameter description:** Specifies the upper limit of the size of the set returned by GIN indexes.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 0

## local\_preload\_libraries

**Parameter description:** Specifies one or more shared libraries that are to be preloaded at connection start. If multiple libraries are to be loaded, separate their names with commas (,). All library names are converted to lower case unless double-quoted.

- Any user can change this option. Therefore, library files that can be loaded are restricted to those saved in the **plugins** subdirectory of the standard library installation directory. It is the database administrator's responsibility to ensure that libraries in this directory are all safe. Entries in **local\_preload\_libraries** can specify the library directory explicitly, for example, **\$libdir/plugins/mylib**, or just specify the library name, for example, **mylib**. (**mylib** is equivalent to **\$libdir/plugins/mylib**.)
- Unlike **shared\_preload\_libraries**, there are no differences in performance between loading a module at session start or doing this during the session. The intent of this feature is to allow debugging or performance-measurement libraries to be loaded into specific sessions without an explicit LOAD command. For example, debugging can be enabled under a given user name by setting this parameter to **ALTER USER SET**.
- If a specified library is not found, the connection attempt will fail.
- Every GaussDB-supported library has a "magic block" that is checked to guarantee compatibility. For this reason, non-GaussDB-supported libraries cannot be loaded in this way.

This parameter is a BACKEND parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** empty

## 14.3.14 Lock Management

In GaussDB, a deadlock may occur when concurrently executed transactions compete for resources. This section describes parameters used for managing transaction locks.

### deadlock\_timeout

**Parameter description:** Specifies the time, in milliseconds, to wait on a lock before checking whether there is a deadlock condition. When the applied lock exceeds the preset value, the system will check whether a deadlock occurs. This parameter takes effect only for common locks.

- The check for deadlock is relatively expensive. Therefore, the server does not check it when waiting for a lock every time. Deadlocks do not frequently occur when the system is running. Therefore, the system just needs to wait on the lock for a while before checking for a deadlock. Increasing this value reduces the time wasted in needless deadlock checks, but slows down reporting of real deadlock errors. On a heavily loaded server, you may need to raise it. The value you have set needs to exceed the transaction time. By doing this, the possibility that a lock will be checked for deadlocks before it is released will be reduced.
- If you want to write the lock wait time during query execution to logs by setting [log\\_lock\\_waits](#), ensure that the value of [log\\_lock\\_waits](#) is less than the specified value (or the default value) of **deadlock\_timeout**.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647. The unit is ms.

**Default value:** 1s

### lockwait\_timeout

**Parameter description:** Specifies the timeout for attempts to acquire a lock. If the time spent in waiting for a lock exceeds the specified time, an error is reported. This parameter takes effect only for common locks.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is ms.

**Default value:** 20 min

### update\_lockwait\_timeout

**Parameter description:** Specifies the maximum duration that a lock waits for concurrent updates on a row to complete when the concurrent update feature is enabled. If the time spent in waiting for a lock exceeds the specified time, an error is reported. This parameter takes effect only for common locks.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is ms.

**Default value:** 120000 (2 min)

## max\_locks\_per\_transaction

**Parameter description:** Specifies the average number of object locks allocated for each transaction.

- The size of the shared lock table is calculated under the condition that a maximum of  $N$  independent objects need to be locked at any time.  $N = \text{max\_locks\_per\_transaction} \times (\text{max\_connections} + \text{max\_prepared\_transactions})$ . Objects whose amount does not exceed the preset number can be locked simultaneously at any time. You may need to increase this value if many different tables are modified in a single transaction. This parameter can only be set at database start.
- Increasing the value of this parameter may cause GaussDB to request more System V-shared memory than the OS's default configuration allows.
- When running a standby server, you must set this parameter to a value that is no less than that on the primary server. Otherwise, queries will not be allowed on the standby server.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10 to 2147483647

**Default value:** 256

## max\_pred\_locks\_per\_transaction

**Parameter description:** Specifies the average number of predicate locks allocated for each transaction.

- The size of the shared predicate lock table is calculated under the condition that a maximum of  $N$  independent objects need to be locked at any time.  $N = \text{max\_pred\_locks\_per\_transaction} \times (\text{max\_connections} + \text{max\_prepared\_transactions})$ . Objects whose amount does not exceed the preset number can be locked simultaneously at any time. You may need to increase this value if many different tables are modified in a single transaction. This parameter can only be set at server start.
- Increasing the value of this parameter may cause GaussDB to request more System V-shared memory than the OS's default configuration allows.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10 to 2147483647

**Default value:** 64

## gs\_clean\_timeout

**Parameter description:** Controls the average interval between **gs\_clean** invocations by the Coordinator.

- Transactions in GaussDB are committed in two phases. An unfinished two-phase transaction may hold a table-level lock, keeping tables from being locked by other connections. In this case, the database needs to invoke the **gs\_clean** tool to clean unfinished two-phase transactions. **gs\_clean\_timeout** is used to control the interval for the Coordinator to invoke the **gs\_clean** tool.
- A larger value of this parameter indicates a low frequency of **gs\_clean** invocation to clean unfinished two-phase transactions.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483. The unit is s.

**Default value:** 1 min

## partition\_lock\_upgrade\_timeout

**Parameter description:** Specifies the timeout for attempts to upgrade an exclusive lock (read allowed) to an access exclusive lock (read/write blocked) on a partitioned table during the execution of some query statements. If there are concurrent read transactions running, the lock upgrade will need to wait. This parameter sets the waiting timeout for lock upgrade attempts.

- When you do **MERGE PARTITION** and **CLUSTER PARTITION** on a partitioned table, temporary tables are used for data rearrangement and file exchange. To concurrently perform as many operations as possible on the partitions, exclusive locks are acquired for the partitions during data rearrangement and access exclusive locks are acquired during file exchange.
- Generally, a partition waits until it acquires a lock, or a timeout occurs if the partition waits for a period longer than the value specified by the **lockwait\_timeout** parameter.
- When doing **MERGE PARTITION** or **CLUSTER PARTITION** on a partitioned table, an access exclusive lock needs to be acquired during file exchange. If the lock fails to be acquired, the acquisition is retried at an interval of 50 ms until timeout occurs. The **partition\_lock\_upgrade\_timeout** parameter specifies the time to wait before the lock acquisition attempt times out.
- If this parameter is set to **-1**, the lock upgrade never times out. The lock upgrade is continuously retried until it succeeds.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 3000. The unit is s.

**Default value:** 1800

## fault\_mon\_timeout

**Parameter description:** Specifies the period for detecting lightweight deadlocks. This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1440. The unit is minute.

**Default value:** 5 min

## enable\_online\_ddl\_waitlock

**Parameter description:** Specifies whether to block DDL operations to wait for the release of cluster locks, such as **pg\_advisory\_lock** and **pgxc\_lock\_for\_backup**. This parameter is mainly used in online OM operations and you are not advised to modify the settings.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** off

## xloginsert\_locks

**Parameter description:** Specifies the number of locks on concurrent write-ahead logging. This parameter is used to improve the efficiency of writing write-ahead logs.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 1000. If the CPU uses the NUMA architecture, the value must be an integer multiple of the number of NUMA nodes.

**Default value:** 16

## num\_internal\_lock\_partitions

**Parameter description:** Specifies the number of internal lightweight lock partitions. It is mainly used for performance optimization in various scenarios. The content is organized in the KV format of keywords and numbers. Different types of locks are separated by commas (.). The sequence does not affect the setting result. For example, **CLOG\_PART=256,CSNLOG\_PART=512** is equivalent to **CSNLOG\_PART=512,CLOG\_PART=256**. If you set the same keyword multiple times, only the latest setting takes effect. For example, if you set **CLOG\_PART** to **256** and **CLOG\_PART** to **2**, the value of **CLOG\_PART** is **2**. If no keyword is set, the default value is used. The usage description, maximum value, minimum value, and default value of each lock type are as follows:

- **CLOG\_PART:** number of Clog file controllers. Increasing the value of this parameter improves the Clog writing efficiency and transaction submission performance, but increases the memory usage. Decreasing the value of this parameter reduces the memory usage, but may increase the conflict of writing Clogs and affect the performance. The value ranges from 1 to 256.
- **CSNLOG\_PART:** number of CSNLOG file controllers. Increasing the value of this parameter improves the CSNLOG log writing efficiency and transaction submission performance, but increases the memory usage. Decreasing the value of this parameter reduces the memory usage, but may increase the conflict of writing CSNLOG logs and affect the performance. The value ranges from 1 to 512.

- **LOG2\_LOCKTABLE\_PART**: two logarithms of the number of common table lock partitions. Increasing the value can improve the concurrency of obtaining locks in the normal process, but may increase the time required for transferring and clearing locks. When wait events occur in **LockMgrLock**, you can increase the value to improve the performance. The minimum value is **4**, indicating that the number of lock partitions is 16. The maximum value is **16**, indicating that the number of lock partitions is 65536.
- **TWOPHASE\_PART**: number of partitions of the two-phase transaction lock. Increasing the value can increase the number of concurrent two-phase transaction commits. The value ranges from 1 to 64.
- **FASTPATH\_PART**: maximum number of locks that each thread can obtain without using the main lock table. When a partitioned table is read, updated, inserted, or deleted and the wait event is LockMgrLock, you can increase the value of this parameter to prevent LockMgrLock from being obtained and improve performance. It is recommended that the value be greater than or equal to that calculated using the following formula: Number of partitions x (1 + Number of local indexes) + Number of global indexes + 10. Increasing the value will increase the memory usage. The value ranges from 20 to 10000.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string.

**Default value:**

- **CLOG\_PART**: 256
- **CSNLOG\_PART**: 512
- **LOG2\_LOCKTABLE\_PART**: 4
- **TWOPHASE\_PART**: 1
- **FASTPATH\_PART**: 20

## enable\_wait\_exclusive\_lock

**Parameter description:** Specifies whether to enable the hang detection and cure function for the exclusive lock of ProcArrayLock.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on**: enabled.
- **off**: disabled.

**Default value:** on

## 14.3.15 Version and Platform Compatibility

### 14.3.15.1 Compatibility with Earlier Versions

This section describes the parameters that control the backward compatibility and external compatibility of GaussDB. A backward compatible database supports

applications of earlier versions. This section describes parameters used for controlling backward compatibility of a database.

## array\_nulls

**Parameter description:** Controls whether the array input parser recognizes unquoted NULL as a null array element.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that null values can be entered in arrays.
- **off** indicates backward compatibility with the old behavior. Arrays containing the value **NULL** can still be created when this parameter is set to **off**.

**Default value:** on

## backslash\_quote

**Parameter description:** Controls whether a single quotation mark can be represented by \ in a string text.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

When the string text meets the SQL standards, \ has no other meanings. This parameter only affects the handling of non-standard-conforming string texts, including escape string syntax (E'...').

---

**Valid value:** enumerated values

- **on** indicates that the use of \ is always allowed.
- **off** indicates that the use of \ is rejected.
- **safe\_encoding** indicates that the use of \ is allowed only when client encoding does not allow ASCII \ within a multibyte character.

**Default value:** safe\_encoding

## default\_with\_oids

**Parameter description:** Specifies whether **CREATE TABLE** and **CREATE TABLE AS** include an **OID** field in newly-created tables if neither **WITH OIDS** nor **WITHOUT OIDS** is specified. It also determines whether OIDs will be included in tables created by **SELECT INTO**.

It is not recommended that OIDs be used in user tables. Therefore, this parameter is set to **off** by default. When OIDs are required for a particular table, **WITH OIDS** needs to be specified during the table creation.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that **CREATE TABLE** and **CREATE TABLE AS** can include an **OID** field in newly-created tables.
- **off** indicates that **CREATE TABLE** and **CREATE TABLE AS** cannot include any **OID** field in newly-created tables.

**Default value:** off

## escape\_string\_warning

**Parameter description:** Specifies whether to issue a warning when a backslash (\) is used as an escape in an ordinary character string.

- Applications that wish to use a backslash (\) as an escape need to be modified to use escape string syntax (E'...'). This is because the default behavior of ordinary character strings treats the backslash as an ordinary character in each SQL standard.
- This variable can be enabled to help locate code lines that need to be changed.
- If E'...' is used as an escape, logs may be incomplete in some scenarios.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**Default value:** on

## lo\_compat\_privileges

**Parameter description:** Specifies whether to enable backward compatibility for the privilege check of large objects.

This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**on** indicates that the privilege check is disabled when users read or modify large objects. This setting is compatible with versions earlier than PostgreSQL 9.0.

**off** indicates that privilege check is enabled for large objects.

**Default value:** off

## quote\_all\_identifiers

**Parameter description:** Specifies whether to forcibly quote all identifiers even if they are not keywords when the database generates SQL. This will affect the output of **EXPLAIN** and the results of functions, such as `pg_get_viewdef`. For details, see the `--quote-all-identifiers` parameter of `gs_dump`.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the forcible quotation is enabled.
- **off** indicates that the forcible quotation is disabled.

**Default value:** off

## sql\_inheritance

**Parameter description:** Controls the inheritance semantics. This parameter specifies the access policy of descendant tables. **off** indicates that subtables cannot be accessed by commands. That is, the ONLY keyword is used by default. It is set for compatibility with versions earlier than PostgreSQL 7.1.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that subtables can be accessed.
- **off** indicates that subtables cannot be accessed.

**Default value:** on

## standard\_conforming\_strings

**Parameter description:** Controls whether ordinary string texts ('...') treat backslashes as ordinary texts as specified in the SQL standard.

- Applications can check this parameter to determine how string texts will be processed.
- It is recommended that characters be escaped by using the escape string syntax (E'...').

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that backslashes are treated as ordinary texts.
- **off** indicates that backslashes are not treated as ordinary texts.

**Default value:** on

## synchronize\_seqscans

**Parameter description:** Controls sequential scans of tables to synchronize with each other, so that concurrent scans read the same data block at about the same time and share the I/O workload.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that a scan may start in the middle of the table and then "wrap around" the end to cover all rows to synchronize with the activity of scans already in progress. This may result in unpredictable changes in the row ordering returned by queries that have no ORDER BY clause.

- **off** indicates that the scan always starts from the table heading.

**Default value:** on

## enable\_beta\_features

**Parameter description:** Specifies whether to enable some features that are not officially released and are used only for POC verification, such as GDS table join. Exercise caution when enabling these extended features because they may cause errors in some scenarios.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the features are enabled for forward compatibility. Note that enabling them may cause errors in certain scenarios.
- **off** indicates that the features are disabled.

**Default value:** off

### 14.3.15.2 Platform and Client Compatibility

Many platforms use the database system. External compatibility of the database system provides significant convenience for platforms.

## transform\_null\_equals

**Parameter description:** Specifies whether expressions of the form `expr = NULL` (or `NULL = expr`) are treated as `expr IS NULL`. They return true if `expr` evaluates to the null value, and false otherwise.

- The correct SQL-standard-compliant behavior of `expr = NULL` is to always return null (unknown).
- Filtered forms in Microsoft Access generate queries that appear to use `expr = NULL` to test for null values. If you turn this option on, you can use this interface to access the database.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that expressions of the form `expr = NULL` (or `NULL = expr`) are treated as `expr IS NULL`.
- **off** indicates that `expr = NULL` always returns null (unknown).

**Default value:** off

#### NOTE

New users are always confused about the semantics of expressions involving **NULL** values. Therefore, **off** is used as the default value.

## support\_extended\_features

**Parameter description:** Specifies whether extended database features are supported.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that extended database features are supported.
- **off** indicates that extended database features are not supported.

**Default value:** off

## lastval\_supported

**Parameter description:** Specifies whether the lastval function can be used.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the lastval function can be used and the nextval function cannot be pushed down.
- **off** indicates that the lastval function cannot be used and the nextval function can be pushed down.

**Default value:** off

## group\_concat\_max\_len

**Parameter description:** This parameter is used together with the **GROUP\_CONCAT** function to limit the length of the return value. If the length exceeds the limit, the exceeded part of the return value is truncated. However, the distributed does not support the **GROUP\_CONCAT** function. Therefore, this function can be set but has no actual effect.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 to 9223372036854775807

**Default value:** 1024

## sql\_compatibility

**Parameter description:** Specifies the type of mainstream database with which the SQL syntax and statement behavior of the database is compatible. This parameter is an INTERNAL parameter. It can be viewed but cannot be modified.

**Value range:** enumerated values

- **ORA** indicates that the SQL syntax and statement behavior of the database is compatible with the Oracle database.
- **TD** indicates that the SQL syntax and statement behavior of the database is compatible with the Teradata database.

- **MYSQL** indicates that the SQL syntax and statement behavior of the database is compatible with the MySQL database.
- **PG** indicates that the database is compatible with the PostgreSQL database.

**Default value:** MYSQL

---

#### NOTICE

- This parameter can be set only when you run the **CREATE DATABASE** command to create a database. For details, see "SQL Reference > SQL Syntax > CREATE DATABASE" in *Developer Guide*.
  - In the database, this parameter must be set to a specific value. It can be set to **ORA** or **TD** and cannot be changed randomly. Otherwise, the setting is not consistent with the database behavior.
- 

## behavior\_compat\_options

**Parameter description:** Specifies database compatibility behavior. Multiple items are separated by commas (,).

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** ""

#### NOTE

- Currently, only items in [Platform and Client Compatibility](#) are supported.
- Multiple items are separated by commas (,), for example, **set behavior\_compat\_options='end\_month\_calculate,display\_leading\_zero'**;

**Table 14-9** Compatibility configuration items

Compatibility Configuration Item	Behavior
display_leading_zero	<p>Specifies how floating point numbers are displayed. It controls the display of zeros before the decimal point of all character string types (such as char, character, nchar, varchar, character varying, varchar2, nvarchar2, text, and clob) and any precision types (such as float4, float8, and numeric) in the numeric type. In addition, the length of the calculated number is displayed synchronously.</p> <ul style="list-style-type: none"> <li>If this item is not specified, for a decimal number between -1 and 1, the 0 before the decimal point is not displayed. For example:  <pre>gaussdb=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d;  a   b   c   d -----+-----+-----+--- .1231243   .1231243   .123   8 (1 row)</pre> </li> <li>If this item is specified, for a decimal number between -1 and 1, the 0 before the decimal point is displayed. For example:  <pre>gaussdb=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d;  a   b   c   d -----+-----+-----+--- 0.1231243   0.1231243   0.123   9 (1 row)</pre> </li> </ul>
end_month_calculate	<p>Specifies the calculation logic of the <b>add_months</b> function. Assume that the two parameters of the <b>add_months</b> function are <b>param1</b> and <b>param2</b>, and that the month of <b>param1</b> and <b>param2</b> is <b>result</b>.</p> <ul style="list-style-type: none"> <li>If this item is not specified, and the <b>Day</b> of <b>param1</b> indicates the last day of a month shorter than <b>result</b>, the <b>Day</b> in the calculation result will equal that in <b>param1</b>. For example:  <pre>gaussdb=# select add_months('2018-02-28',3) from sys_dummy, add_months ----- 2018-05-28 00:00:00 (1 row)</pre> </li> <li>If this item is specified, and the <b>Day</b> of <b>param1</b> indicates the last day of a month shorter than <b>result</b>, the <b>Day</b> in the calculation result will equal that in <b>result</b>. For example:  <pre>gaussdb=# select add_months('2018-02-28',3) from sys_dummy, add_months ----- 2018-05-31 00:00:00 (1 row)</pre> </li> </ul>

Compatibility Configuration Item	Behavior
compat_analyze_sample	<p>Specifies the sampling behavior of the ANALYZE operation. If this item is specified, the sample collected by the ANALYZE operation will be limited to around 30,000 records, controlling CN memory consumption and maintaining the stability of ANALYZE.</p>
bind_schema_tablespace	<p>Binds a schema with the tablespace with the same name. If a tablespace name is the same as <i>sche_name</i>, <b>default_tablespace</b> will also be set to <i>sche_name</i> if <b>search_path</b> is set to <i>sche_name</i>.</p>
bind_procedure_searchpath	<p>Specifies the search path of the database object in a stored procedure for which no schema name is specified. If no schema name is specified for a stored procedure, the search is performed in the schema to which the stored procedure belongs. If the stored procedure is not found, the following operations are performed:</p> <ul style="list-style-type: none"> <li>• If this item is not specified, the system reports an error and exits.</li> <li>• If this item is specified, the search continues based on the settings of <b>search_path</b>. If the issue persists, the system reports an error and exits.</li> </ul>
correct_to_number	<p>Controls the compatibility of the <b>to_number()</b> result. If this item is not set, the result of the to_number() function is the same as that in the ORA database by default.</p> <pre>gaussdb=# select " AS to_number_14, to_number('34,50','999,99'); ERROR: invalid data. CONTEXT: referenced column: to_number</pre> <p>If this item is set, the result of the to_number() function is the same as that of pg11.</p> <pre>gaussdb=# select " AS to_number_14, to_number('34,50','999,99'); to_number_14   to_number -----+-----   3450 (1 row)</pre>

Compatibility Configuration Item	Behavior
<p>unbind_divide_bound</p>	<p>Controls the range check on the result of integer division. If this item is not set, the range of the division result is verified. For example, an out-of-bounds error is reported because the output result of <i>INT_MIN/(-1)</i> is greater than <i>INT_MAX</i>.</p> <pre>gaussdb=# select (-2147483648)::int4 / (-1)::int4; ERROR: integer out of range</pre> <p>If this item is set, the range of the division result does not need to be verified. For example, the output result of <i>INT_MIN/(-1)</i> is <i>INT_MAX+1</i>.</p> <pre>gaussdb=# select (-2147483648)::int4 / (-1)::int4; ?column? ----- 2147483648 (1 row)</pre>
<p>convert_string_digit_to_numeric</p>	<p>Determines whether to convert numeric constants of the character string type to those of the numeric type before these two types are compared.</p> <pre>gaussdb=# create table test1 (c1 int, c2 varchar); gaussdb=# insert into test1 values (2, '1.1'); gaussdb=# set behavior_compat_options=""; gaussdb=# select * from test1 where c2 &gt; 1; ERROR: invalid input syntax for type bigint: "1.1"</pre> <pre>gaussdb=# set behavior_compat_options='convert_string_digit_to_numeric'; gaussdb=# select * from test1 where c2 &gt; 1;  c1   c2 ----+----   2   1.1 (1 row)</pre>
<p>return_null_string</p>	<p>Specifies how to display the empty result (empty string ") of the lpad() and rpad() functions.</p> <ul style="list-style-type: none"> <li>If this item is not specified, the empty string is displayed as <b>NULL</b>.</li> </ul> <pre>gaussdb=# select length(lpad('123',0,'*')) from sys_dummy, length ----- (1 row)</pre> <ul style="list-style-type: none"> <li>If this item is specified, the empty string is displayed as single quotation marks (").</li> </ul> <pre>gaussdb=# select length(lpad('123',0,'*')) from sys_dummy, length ----- 0 (1 row)</pre>

Compatibility Configuration Item	Behavior
<p>compat_concat_variadic</p>	<p>Specifies the compatibility of variadic results of the <b>concat()</b> and <b>concat_ws()</b> functions. The MySQL database does not have the variadic type. Therefore, this option has no impact on the MY database.</p> <p>If this item is not set and the concat function parameter is of the variadic type, the results of the ORA and TD databases in compatibility mode are the same by default.</p> <pre>gaussdb=# select concat(variadic NULL::int[]) is NULL; ?column? ----- t (1 row)</pre> <p>If this item is set and the concat function parameter is of the variadic type, different result formats of the ORA and TD databases in compatibility mode are retained.</p> <p>-- In the ORA database:</p> <pre>gaussdb=# select concat(variadic NULL::int[]) is NULL; ?column? ----- t (1 row)</pre> <p>-- In the TD database:</p> <pre>gaussdb=# select concat(variadic NULL::int[]) is NULL; ?column? ----- f (1 row)</pre>
<p>merge_update_multi</p>	<p>When MERGE INTO ... WHEN MATCHED THEN UPDATE (see "SQL Reference &gt; SQL Syntax &gt; MERGE INTO" in <i>Developer Guide</i>) and INSERT ... ON DUPLICATE KEY UPDATE (see "SQL Reference &gt; SQL Syntax &gt; INSERT" in <i>Developer Guide</i>) are used, it controls the UPDATE behavior if a piece of target data in the target table conflicts with multiple pieces of source data.</p> <p>If this item is specified and the preceding scenario exists, the system performs multiple UPDATE operations on the conflicting row. If this item is not specified and the preceding scenario exists, an error is reported, that is, the MERGE or INSERT operation fails.</p>
<p>plstmt_implicit_savepoint</p>	<p>Determines whether the execution of an UPDATE statement in a stored procedure has an independent subtransaction.</p> <p>If this parameter is set, the implicit savepoint is enabled before executing each UPDATE statement in the stored procedure, and the subtransaction is rolled back to the latest savepoint in the EXCEPTION block by default, ensuring that only the modification of failed statements is rolled back. This option is used to be compatible with the <b>EXCEPTION</b> behavior of the O database.</p>

Compatibility Configuration Item	Behavior
hide_tailing_zero	<p>Configuration item for numeric display. If this parameter is not set, numeric values are displayed based on the specified precision. If this parameter is set, all numeric values are output with trailing zeros (after a decimal point) hidden, including the to_char(numeric, format) scenario.</p> <p>For example:</p> <pre>gaussdb=# set behavior_compat_options='hide_tailing_zero'; gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999');  a   to_char -----+----- 123.123   123.123 (1 row) gaussdb=# set behavior_compat_options=""; gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999');  a   to_char -----+----- 123.1230000000   123.123000 (1 row)</pre>
plsql_security_definer	<p>After this parameter is enabled, the definer permission is used by default when a stored procedure is created.</p>

Compatibility Configuration Item	Behavior
char_coerce_compat	<p>Specifies the behavior when the char(n) type is converted to other variable-length string types. If this parameter is not set, spaces at the end are omitted when the char(n) type is converted to other variable-length string types. If this parameter is set, spaces at the end are not omitted during conversion. In addition, if the length of the char(n) type exceeds the length of other variable-length string types, an error is reported. This parameter is valid only when the <b>sql_compatibility</b> parameter is set to <b>ORA</b>. After this parameter is enabled, spaces at the end are not omitted in implicit conversion, explicit conversion, or conversion by calling the <b>text(bpchar)</b> function.</p> <pre> gaussdb=# set behavior_compat_options=""; gaussdb=# create table tab_1(col1 varchar(3)); gaussdb=# create table tab_2(col2 char(3)); gaussdb=# insert into tab_2 values(' '); gaussdb=# insert into tab_1 select col2 from tab_2; gaussdb=# select * from tab_1 where col1 is null; col1 ----- (1 row) gaussdb=# select * from tab_1 where col1=' '; col1 ----- (0 rows) gaussdb=# delete from tab_1; gaussdb=# set behavior_compat_options = 'char_coerce_compat'; gaussdb=# insert into tab_1 select col2 from tab_2; gaussdb=# select * from tab_1 where col1 is null; col1 ----- (0 rows) gaussdb=# select * from tab_1 where col1=' '; col1 ----- (1 row) </pre>
truncate_numeric_tail_zero	<p>Configuration item for numeric display. If this parameter is not set, numeric values are displayed based on the default precision. If this parameter is set, all numeric values are output with trailing zeros (after a decimal point) hidden, except for to_char(numeric, format). For example:</p> <pre> gaussdb=# set behavior_compat_options='truncate_numeric_tail_zero'; gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a   to_char -----+----- 123.123   123.123000 (1 row) gaussdb=# set behavior_compat_options=""; gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a   to_char -----+----- 123.1230000000   123.123000 (1 row) </pre>

Compatibility Configuration Item	Behavior
plpgsql_dependency	<p>If this parameter is set, a function or stored procedure containing undefined objects can be created. You can query the dependency in <b>GS_DEPENDENCIES</b> and <b>GS_DEPENDENCIES_OBJ</b>.</p> <p>If this parameter is enabled, when creating a PL/SQL object, the OID that depends on the PL/SQL object is automatically updated.</p> <p>Dependency can be established in the following scenarios:</p> <ol style="list-style-type: none"> <li>1. A function appears at the position of the dependency type and parameter default value in a function header.</li> <li>2. Type in the function and variable dependency type.</li> <li>3. In the function body, function A is called in the right value expression of the function call or assignment statement. If the input and output parameters of function A contain function B, no dependency is established for function B. For example, <b>functionA(functionB())</b>. Only the dependency for function A is established.</li> <li>4. Dependency function in a view.</li> </ol> <p>Dependency cannot be established in the following scenarios:</p> <ol style="list-style-type: none"> <li>1. A type in a schema depends on other types.</li> <li>2. Dependency on functions, variables, tables, and views in SQL statements. For example, dependency is not recorded for <b>select id into var1 from table1 join view1 on table1.id = pkg1.var1; table1,view1,pkg1</b>.</li> <li>3. Dependency on functions, variables, tables, and views in a view.</li> </ol> <p>Notes:</p> <ol style="list-style-type: none"> <li>1. When PL/SQL objects are concurrently created, deadlocks may occur if competition occurs among the object.</li> <li>2. If the objects to be modified exist in <b>gs_dependencies</b> and <b>gs_dependencies_obj</b>, you cannot rename the objects.</li> <li>3. When a function, stored procedure, or package depends on a synonym, the synonym must be created in advance. OIDs cannot be maintained manually.</li> </ol>
disable_rewrite_nesttable	The distributed mode does not support this parameter.

Compatibility Configuration Item	Behavior
proc_outparam_override	<p>Determines the overloading of output parameters of a stored procedure. After this parameter is enabled, the stored procedure can be properly created and invoked even if only the output parameters of the stored procedure are different. Currently, this parameter can be used only when gsql and JDBC are used to connect to the database. If this parameter is enabled for other tools to connect to the database, stored procedures with the <b>out</b> parameter cannot be invoked.</p> <p>It supports the functions that contain the <b>out</b> output parameter and returns data of the record type. Besides, a value is assigned to the <b>out</b> parameter.</p>
aformat_regexp_match	<p>Determines the matching behavior of regular expression functions.</p> <p>When this parameter is set and <b>sql_compatibility</b> is set to <b>ORA</b> or <b>MYSQL</b>, the options supported by the <b>flags</b> parameter of the regular expression are changed as follows:</p> <ol style="list-style-type: none"> <li>1. By default, the character '\n' cannot be matched.</li> <li>2. When <b>flags</b> contains the <b>n</b> option, the character '\n' can be matched.</li> <li>3. The <b>regexp_replace(source, pattern replacement)</b> function replaces all matching substrings.</li> <li>4. <b>regexp_replace(source, pattern, replacement, flags)</b> returns null when the value of <b>flags</b> is '' or null.</li> </ol> <p>Otherwise, the meanings of the options supported by the <b>flags</b> parameter of the regular expression are as follows:</p> <ol style="list-style-type: none"> <li>1. By default, the character '\n' can be matched.</li> <li>2. The <b>n</b> option in <b>flags</b> indicates that the multi-line matching mode is used.</li> <li>3. The <b>regexp_replace(source, pattern replacement)</b> function replaces only the first matched substring.</li> <li>4. If the value of <b>flags</b> is '' or null, the return value of <b>regexp_replace(source, pattern, replacement, flags)</b> is the character string after replacement.</li> </ol>
disable_emptystr2null	<p>If this parameter is enabled, the function of converting empty strings to null by default is disabled for the following character types: text, clob, blob, raw, bytea, varchar, nvarchar2, bpchar, char, name, byteawithoutorderwithqualcol, and byteawithoutordercol. This parameter is reserved for emergency. Do not set it unless necessary.</p>
select_into_return_null	<p>This parameter is invalid in distributed mode.</p>

Compatibility Configuration Item	Behavior
proc_uncheck_default_param	<p>When a function is called, the system does not check whether the default parameter is omitted.</p> <ul style="list-style-type: none"> <li>If this item is not set and a function with default parameters is invoked, input parameters are added to the function from left to right. If inputs of non-default parameters are missing, an error is reported. For example: <pre> gaussdb=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int gaussdb=# as gaussdb\$\$ begin gaussdb\$\$ raise info 'f1:%',f1; gaussdb\$\$ raise info 'f2:%',f2; gaussdb\$\$ raise info 'f3:%',f3; gaussdb\$\$ raise info 'f4:%',f4; gaussdb\$\$ raise info 'f5:%',f5; gaussdb\$\$ return 1; gaussdb\$\$ end; gaussdb\$\$ / CREATE FUNCTION gaussdb=# select test(1,2); ERROR: function test(integer, integer) does not exist LINE 1: select test(1,2);                 ^ HINT: No function matches the given name and argument types. You might need to add explicit type casts. CONTEXT: referenced column: test </pre> </li> <li>If this item is set and a function with default parameters is invoked, input parameters are added to the function from left to right. The number of defaulted inputs depends on the number of default parameters. If an input of a non-default parameter is missing, the previous default value is used to fill this parameter. For example: <pre> gaussdb=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int gaussdb=# as gaussdb\$\$ begin gaussdb\$\$ raise info 'f1:%',f1; gaussdb\$\$ raise info 'f2:%',f2; gaussdb\$\$ raise info 'f3:%',f3; gaussdb\$\$ raise info 'f4:%',f4; gaussdb\$\$ raise info 'f5:%',f5; gaussdb\$\$ return 1; gaussdb\$\$ end; gaussdb\$\$ / CREATE FUNCTION gaussdb=# select test(1,2); INFO: f1:1 CONTEXT: referenced column: test INFO: f2:2 CONTEXT: referenced column: test INFO: f3:20 CONTEXT: referenced column: test INFO: f4:40 CONTEXT: referenced column: test INFO: f5:50 CONTEXT: referenced column: test test </pre> </li> </ul>

Compatibility Configuration Item	Behavior
	<p>----- 1 (1 row)</p> <p>As shown above, f3 is filled with an incorrect default value.</p> <p><b>WARNING</b> In this scenario, a non-default parameter is filled with the previous default value.</p>
dynamic_sql_com pat	<p>After this parameter is enabled, the dynamic statement does not consider the template parameters with the same name in the template SQL statement as the same variable. Instead, the variables in the USING clause are matched in sequence.</p> <p><b>CAUTION</b></p> <ul style="list-style-type: none"> <li>• If a stored procedure is invoked when a dynamic statement executes an anonymous block statement, only the <b>IN</b> parameters are corrected. If <b>OUT</b> parameters need to be checked, set the <b>proc_outparam_override</b> parameter.</li> <li>• If a stored procedure is invoked when a dynamic statement executes an anonymous block statement and the parameter is enabled, the <b>IN</b> and <b>OUT</b> attributes in a stored procedure and the USING clause are not checked.</li> </ul>
dynamic_sql_chec k	<p>If this parameter is enabled, an error is reported during dynamic statement execution if the number of different template parameters in the dynamic statement template SQL is different from the number of variables in the USING clause.</p> <p><b>CAUTION</b></p> <ul style="list-style-type: none"> <li>• If the <b>dynamic_sql_compat</b> option is enabled, the <b>dynamic_sql_check</b> option does not take effect.</li> <li>• If a stored procedure is invoked when a dynamic statement executes an anonymous block statement, only the <b>IN</b> parameters are checked. If <b>OUT</b> parameters need to be checked, set the <b>proc_outparam_override</b> parameter.</li> <li>• If a stored procedure is invoked when a dynamic statement executes an anonymous block statement and the parameter is enabled, the <b>IN</b> and <b>OUT</b> attributes in a stored procedure and the USING clause are not checked.</li> </ul>

Compatibility Configuration Item	Behavior
enable_funcname_with_argsname	<p>If the parameter is enabled, the projection alias displays the complete function when the SELECT function is invoked.</p> <ul style="list-style-type: none"> <li> <p>If this item is not set, the projection alias displays the function name when the SELECT function is invoked. For example:</p> <pre>gaussdb=# SELECT power(2,3); power -----       8 (1 row)  gaussdb=# SELECT count(*) FROM db_ind_columns; count -----    611 (1 row)  gaussdb=# SELECT count(index_name) FROM db_ind_columns; count -----    611 (1 row)  gaussdb=# SELECT left('abcde', 2); left -----   ab (1 row)  gaussdb=# SELECT pg_client_encoding(); pg_client_encoding -----       UTF8 (1 row)</pre> </li> <li> <p>If the item is set, the projection alias displays the complete function when the SELECT function is invoked. For example:</p> <pre>gaussdb=# SET behavior_compat_options = 'enable_funcname_with_argsname'; SET gaussdb=# SELECT power(2,3); power(2,3) -----       8 (1 row)  gaussdb=# SELECT count(*) FROM db_ind_columns; count(*) -----    611 (1 row)  gaussdb=# SELECT count(index_name) FROM db_ind_columns; count(index_name) -----           611 (1 row)  gaussdb=# SELECT left('abcde', 2); left('abcde',2)</pre> </li> </ul>

Compatibility Configuration Item	Behavior
	<pre> ----- ab (1 row)  gaussdb=# SELECT pg_client_encoding(); pg_client_encoding() ----- UTF8 (1 row) </pre> <p><b>CAUTION</b></p> <ul style="list-style-type: none"> <li>• Currently, only <b>func_name(args_list)</b>, <b>func_name()</b>, and <b>func_name(*)</b> are supported to display complete functions. The <b>args</b> parameter only supports character type, value type, column name, and function. A function name can contain a schema or package name. The parameter cannot contain other clauses (such as the ORDER BY clause) or be an expression. The parameter can contain only the DISTINCT keyword. If the parameter contains other keywords, the complete function cannot be displayed.</li> <li>• Some special functions do not support displaying projection alias, including COLLATION FOR, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, DBTIMEZONE, LOCALTIME, LOCALTIMESTAMP, SYSDATE, SESSIONTIMEZONE, ROWNUM, CURRENT_ROLE, CURRENT_USER, SESSION_USER, USER, CURRENT_CATALOG, CURRENT_SCHEMA, CAST, EXTRACT, TIMESTAMPDIFF, OVERLAY, POSITION, SUBSTRING, TREAT, TRIM, NULLIF, NVL, NVL2, COALESCE, GREATEST, LEAST, LNNVL, REGEXP_LIKE, and XML functions.</li> <li>• Some security encryption and decryption functions, anonymization functions, and projection aliases display functions may have security problems. Therefore, only function names are displayed here, including gs_encrypt_aes128, gs_decrypt_aes128, gs_encrypt, gs_decrypt, aes_encrypt, aes_decrypt, pg_create_physical_replication_slot_extern, dblink_connect,creditcardmasking, "basicemailmasking", fullemailmasking, alldigitsmasking, shufflemasking, randommasking, regexprmasking, and gs_digest.</li> <li>• Parameter transfer using =&gt; is not supported for the projection alias to display the complete function. The projection alias cannot contain double quotation marks (""), for example, <b>select "power"(2,3)</b>.</li> <li>• To enable the projection alias to display the complete function, this function is not affected by parameters such as removing 0 at the end.</li> </ul>

Compatibility Configuration Item	Behavior
allow_rownum_alias	<p>After this parameter is enabled, ROWNUM can be used as a column alias in SQL statements using the AS syntax. ROWNUM is used as a common identifier and cannot be used as a pseudocolumn.</p> <p><b>WARNING</b> You are not advised to change the status of this parameter during service execution. When the parameter is enabled, database objects (such as table names, column names, and database names) created using ROWNUM as the name in the database can be used only when the parameter is enabled. Otherwise, ambiguity occurs and the behavior is unpredictable. When the parameter is disabled, the behavior of using ROWNUM as a pseudocolumn in the database becomes invalid after the parameter is enabled and the behavior is unpredictable.</p>
current_sysdate	<p>If this parameter is enabled, the current OS time is returned when the <b>sysdate</b> command is executed.</p> <pre>gaussdb=# set behavior_compat_options='current_sysdate'; SET gaussdb=# select sysdate; current_sysdate ----- 2023-06-20 20:15:27 (1 row)</pre>

## a\_format\_version

**Parameter description:** Specifies the database platform compatibility configuration item. The value of this parameter is an enumerated string.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** ""

### NOTICE

- Currently, only compatibility configuration items in [Table 14-10](#) are supported.
- Set a character string for the compatibility configuration item, for example, **set a\_format\_version='10c'**.

**Table 14-10** Compatibility configuration items

Compatibility Configuration Item	Behavior
10c	The ORA platform is compatible with the 10C version.

## a\_format\_dev\_version

**Parameter description:** Specifies the database platform minor version compatibility configuration item. The value of this parameter is an enumerated string.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** ""

### NOTICE

- Currently, only compatibility configuration items in [Table 14-11](#) are supported.
- Set a character string for the compatibility configuration item, for example, **set a\_format\_dev\_version='s1'**.

**Table 14-11** Compatibility configuration items

Compatibility Configuration Item	Behavior
s1	<ul style="list-style-type: none"> <li>• Compatible minor version of the ORA platform, which affects functions TRUNC(date, fmt), ROUND(date, fmt), NVL2, LPAD, RPAD, ADD_MONTHS, MONTHS_BETWEEN, REGEXP_REPLACE, REGEXP_COUNT, TREAT, EMPTY_CLOB, INSTRB, trunc(number), greatest, least, mod, round(number), cast, to_date, to_timestamp, chr, rtrim, translate, to_char, to_number, and to_timestamp_tz.</li> <li>• Data type conversion: A decimal character string is rounded off when it is converted to an integer (int1/int2/int4/int8/int16).</li> <li>• Data type conversion: Implicit conversion from timestamp with time zone to timestamp without time zone is supported.</li> </ul>

Compatibility Configuration Item	Behavior
s2	<ul style="list-style-type: none"> <li>Compatible minor version of platform A, which affects functions, such as <code>dump</code>, <code>to_single_byte</code>, <code>to_multi_byte</code>, <code>nls_upper</code>, <code>nls_lower</code>, <code>initcap</code>, <code>ascii2</code>, <code>asciistr</code>, <code>unistr</code>, <code>vsize</code>, <code>cosh</code>, <code>remainder</code>, <code>sinh</code>, <code>tanh</code>, <code>nanvl</code>, <code>current_date</code>, <code>current_timestamp</code>, <code>dbtimezone</code>, <code>numtodsinterval</code>, <code>numtoyminterval</code>, <code>new_time</code>, <code>sessiontimezone</code>, <code>sys_extract_utc</code>, <code>tz_offset</code>, <code>to_binary_double</code>, <code>to_binary_float</code>, <code>to_dsinterval</code>, <code>to_yminterval</code>, <code>lnnvl</code>, and <code>ora_hash</code>.</li> <li>Supports all behaviors when the compatibility configuration item is set to <b>s1</b>.</li> </ul>

### plpgsql.variable\_conflict

**Parameter description:** Sets the priority of using stored procedure variables and table columns with the same name.

This is a USERSET parameter. Set it based on method 3 provided in [Table 14-1](#).

**Value range:** a string

- error** indicates that a compilation error is reported when the name of a stored procedure variable is the same as that of a table column.
- use\_variable** indicates that if the name of a stored procedure variable is the same as that of a table column, the variable is used preferentially.
- use\_column** indicates that if the name of a stored procedure variable is the same as that of a table column, the column name is used preferentially.

**Default value:** error

### td\_compatible\_truncation

**Parameter description:** Specifies whether to enable features compatible with a Teradata database. You can set this parameter to **on** when connecting to a database compatible with the Teradata database, so that when you perform the INSERT operation, overlong strings are truncated based on the allowed maximum length before being inserted into char- and varchar-type columns in the target table. This ensures all data is inserted into the target table without errors reported.

 NOTE

The string truncation function cannot be used if the INSERT statement includes a foreign table.

If inserting multi-byte character data (such as Chinese characters) to database with the character set byte encoding (such as SQL\_ASCII or LATIN1), and the character data crosses the truncation position, the string is truncated based on its bytes instead of characters. Unexpected result will occur in tail after the truncation. If you want correct truncation result, you are advised to adopt encoding set such as UTF8, which has no character data crossing the truncation position.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that overlong strings are truncated.
- **off** indicates that overlong strings are not truncated.

**Default value:** off

## nls\_timestamp\_format

**Parameter description:** Specifies the default timestamp format.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** DD-Mon-YYYY HH:MI:SS.FF AM

## nls\_timestamp\_tz\_format

**Parameter description:** Specifies the default timestamp with time zone format.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. The supported formats are the same as those of `nls_timestamp_format`.

**Default value:** DD-Mon-YYYY HH:MI:SS.FF AM

 NOTE

This parameter is valid only when `a_format_version` is set to **10c** and `a_format_dev_version` is set to **s1**.

## max\_function\_args

**Parameter description:** Specifies the maximum number of parameters allowed for a function.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Value range:** an integer.

**Default value:** 8192

## convert\_string\_to\_digit

**Parameter description:** Specifies the implicit conversion priority, which determines whether to preferentially convert strings into numbers.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that strings are preferentially converted into numbers.
- **off** indicates that strings are not preferentially converted into numbers.

**Default value:** on

---

### NOTICE

Adjusting this parameter will change the internal data type conversion rule and cause unexpected behavior. Exercise caution when performing this operation.

---

## 14.3.16 Fault Tolerance

This section describes parameters used for controlling how the server processes an error occurring in the database system.

### exit\_on\_error

**Parameter description:** If this function is enabled, errors of the ERROR level will be upgraded to PANIC errors, and core stacks will be generated. It is mainly used to locate problems and test services.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that errors of the ERROR level will be upgraded to PANIC errors.
- **off** indicates that errors of the ERROR level will not be upgraded.

**Default value:** off

### restart\_after\_crash

**Parameter description:** If this parameter is set to **on** and a backend process crashes, GaussDB automatically reinitializes the backend process.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** maximizes the availability of the database.  
In some circumstances (for example, when a management tool, such as xCAT, is used to manage GaussDB), setting this parameter to **on** maximizes the availability of the database.

- **off** indicates that a management tool is enabled to obtain control permission and take proper measures when a backend process crashes.

**Default value:** on

## omit\_encoding\_error

**Parameter description:** If this parameter is set to **on** and the client character set of the database is encoded in UTF-8 format, character encoding conversion errors will be recorded in logs. Additionally, converted characters that have conversion errors will be ignored and replaced with question marks (?).

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that characters that have conversion errors will be ignored and replaced with question marks (?), and error information will be recorded in logs.
- **off** indicates that characters that have conversion errors cannot be converted and error information will be directly displayed.

**Default value:** off

### NOTE

If this parameter is modified by running the **gs\_guc reload** command and the connection of a session on the current node is not from the client but from another node in the cluster to which the node belongs, this parameter does not take effect immediately on the session after the **gs\_guc reload** command is executed. The setting takes effect only after the connection node is disconnected and then reconnected.

## cn\_send\_buffer\_size

**Parameter description:** Specifies the size of the data buffer used for data transmission on CNs.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 8 to 128. The unit is KB.

**Default value:** 8 KB

## data\_sync\_retry

**Parameter description:** Specifies whether to keep running the database when updated data fails to be written into disks by using the **fsync** function. In some OSs, no error is reported even if **fsync** fails after the second attempt. As a result, data is lost.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that the database keeps running and **fsync** is executed again after **fsync** fails.
- **off** indicates that a PANIC-level error is reported and the database is stopped after **fsync** fails.

**Default value:** off

## remote\_read\_mode

**Parameter description:** Specifies whether to enable the remote read function. This function allows pages on the standby server to be read when reading pages on the primary server fails.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **off** indicates that the remote read function is disabled.
- **non\_authentication** indicates that the remote read function is enabled but certificate authentication is not required.
- **authentication** indicates that the remote read function is enabled and certificate authentication is required.

**Default value:** authentication

## 14.3.17 Connection Pool Parameters

When a connection pool is used to access the database, database connections are established and then stored in the memory as objects during system running. When you need to access the database, no new connection is established. Instead, an existing idle connection is selected from the connection pool. After you finish accessing the database, the database does not disable the connection but puts it back into the connection pool. The connection can be used for the next access request.

### pooler\_port

**Parameter description:** Specifies the O&M management port of internal tools, such as **cm\_agent** and **cm\_ctl**. This port is used by the initial user or system administrator to connect to the database through the client.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** the value of the GUC parameter **port** of a CN or DN plus 1

**Default value:** the default value of the GUC parameter **port** of a CN or DN plus 1. The default value of this parameter is **8001** for CNs and **40001** for DNs.

### pooler\_maximum\_idle\_time

**Parameter description:** Specifies the maximum amount of time that the connections can remain idle in a pool before being removed. After that, the automatic connection clearing mechanism is triggered to reduce the number of connections on each node to the value of **minimum\_pool\_size**.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 2147483647. The smallest unit is s.

**Default value:** 10 min (600s)

## minimum\_pool\_size

**Parameter description:** Specifies the minimum number of remaining connections in the pool on each node after the automatic connection clearing is triggered. If this parameter is set to **0**, the automatic connection clearing is disabled.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 65535.

**Default value:** 50

## max\_pool\_size

**Parameter description:** Specifies the maximum number of connections between a CN and another CN/DN in a connection pool. This parameter may need to be modified when the cluster scale changes. For example, nodes are added or deleted.

**Parameter type:** integer

**Unit:** none

**Value range:** 1 to 65535

**Default value:**

- Independent deployment:  
**32768** (60-core CPU/480 GB memory); **16384** (32-core CPU/256 GB memory);  
**8192** (16-core CPU/128 GB memory); **4096** (8-core CPU/64 GB memory);  
**2048** (4-core CPU/32 GB memory); **1000** (4-core CPU/16 GB memory)

**Setting method:** This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Setting suggestion:** Set this parameter based on the default value of different instance specifications. The value of this parameter must be greater than the value of **max\_connections**. During the adjustment, the connections consumed by internal threads must be reserved. When the number of concurrent services is large, the connections from the CN to other CNs/DNs in the connection pool are consumed. If this parameter is set to a small value, an error is reported when the number of connections reaches the upper limit. As a result, services fail. When a CN is started, it allocates memory in advance based on the parameter value. Therefore, if the parameter value increases, the system consumes more memory resources. However, in general, the impact on the CN memory is small.

## persistent\_datanode\_connections

**Parameter description:** Specifies whether to release the connection for the current session.

This parameter is a BACKEND parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **off** indicates that the connection for the current session will be released.
- **on** indicates that the connection for the current session will not be released.

---

**NOTICE**

After this parameter is set to **on**, a session may hold a connection but does not run a query. As a result, other query requests fail to be connected. To fix this problem, the number of sessions must be less than or equal to **max\_active\_statements**.

---

**Default value:** off

## max\_coordinators

**Parameter description:** Specifies the maximum number of CNs in a cluster. This parameter may need to be modified when the cluster scale changes, for example, nodes are added or deleted. During scale-out, ensure that the value of this parameter is greater than the number of CNs in the target cluster. Otherwise, the scale-out will fail. If this parameter is set to a large value during scale-in, more memory resources are consumed when the CN is started. However, in general, the impact on the CN memory is small.

**Parameter type:** integer

**Unit:** none

**Value range:** 2 to 1024

**Default value:** 128

**Setting method:** This is a POSTMASTER parameter. You are not advised to modify it. If you need to modify it, set it based on instructions provided in [Table 14-2](#).

**Setting suggestion:** Set this parameter based on the actual cluster specifications. If the value of this parameter is less than the number of CNs in the cluster, node creation will fail. When a CN is started, it allocates memory in advance based on the parameter value. Therefore, if the parameter value increases, the system consumes more memory resources. However, in general, the impact on the CN memory is small.

## max\_datanodes

**Parameter description:** Specifies the maximum number of DN nodes in a cluster. This parameter may need to be modified when the cluster scale changes, for example, nodes are added or deleted. During scale-out, ensure that the value of this parameter is greater than the total number of DN shards in the target cluster. Otherwise, the scale-out will fail. If this parameter is set to a large value during scale-in, more memory resources are consumed when the CN is started. However, in general, the impact on the CN memory is small.

**Parameter type:** integer

**Unit:** none

**Value range:** 2 to 65535

**Default value:** 256

**Setting method:** This is a POSTMASTER parameter. You are not advised to modify it. If you need to modify it, set it based on instructions provided in [Table 14-2](#).

**Setting suggestion:** Set this parameter based on the actual cluster specifications. If the value of this parameter is less than the number of DNs in the cluster, node creation will fail. When a CN is started, it allocates memory in advance based on the parameter value. Therefore, if the parameter value increases, the system consumes more memory resources. However, in general, the impact on the CN memory is small.

## cache\_connection

**Parameter description:** Specifies whether to reclaim the connections of a connection pool.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that the connections of a connection pool will be reclaimed.
- **off** indicates that the connections of a connection pool will not be reclaimed.

**Default value:** on

## enable\_force\_reuse\_connections

**Parameter description:** Specifies whether a session forcibly reuses a new connection.

This parameter is a BACKEND parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that the new connection is forcibly used.
- **off** indicates that the current connection is used.

**Default value:** off

## pooler\_connect\_max\_loops

**Parameter description:** Specifies whether to enable the connection retries to enhance stability of setting up connections in primary/standby switchover scenarios. If a service fails to connect to the primary server, it will retry by attempting to connect to the standby server. If the standby server is successfully promoted to primary, the retry attempt will succeed. This parameter specifies the total number of retry attempts. If this parameter is set to **0**, retries are disabled. The service only establishes a connection to the primary server.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 20.

**Default value:** 1

## pooler\_connect\_interval\_time

**Parameter description:** Specifies the interval between retries when **pooler\_connect\_max\_loops** is set to a value greater than 1. You are advised to set this parameter to a value slightly greater than the time required for primary/standby switchover in the current cluster.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 7200. The smallest unit is s.

**Default value:** 15s

## pooler\_timeout

**Parameter description:** Specifies the timeout period of communication between each connection in a CN's connection pool and another CN/DN.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 7200. The smallest unit is s.

**Default value:** 10 min

## pooler\_connect\_timeout

**Parameter description:** Specifies the timeout period of connecting a CN's connection pool to another CN/DN in the same cluster.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 7200. The smallest unit is s.

**Default value:** 1 min

## pooler\_cancel\_timeout

**Parameter description:** Specifies the timeout period of canceling a connection by a CN's connection pool during error processing. If similar timeout occurs when an exception of the subtransaction or stored procedure is captured, the transaction containing the subtransaction or the stored procedure rolls back. If the source data from the COPY FROM operation is not consistent with that of the table structure in the target table, and the parameter value is not 0, an error is reported.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 7200. The smallest unit is s. **0** (not recommended) indicates that the timeout is disabled.

**Default value:** 15s

## 14.3.18 Cluster Transaction Parameters

This section describes the settings and value ranges of transaction parameters for the cluster.

### transaction\_isolation

**Parameter description:** specifies the isolation level of the current transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string of case-sensitive characters. The values include:

- **serializable:** This value is equivalent to REPEATABLE READ in GaussDB.
- **read committed** indicates that only the data in committed transactions will be read.
- **repeatable read** indicates that only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **read uncommitted** indicates that data is readable at any time.
- **default:** The value is the same as that of **default\_transaction\_isolation**.

**Default value:** read committed

### transaction\_read\_only

**Parameter description:** Specifies whether the current transaction is a read-only transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that the current transaction is a read-only transaction.
- **off** indicates that the current transaction can be a read/write transaction.

**Default value:** off

### xc\_maintenance\_mode

**Parameter description:** Specifies whether the system is in maintenance mode.

This parameter is a SUSERSET parameter. Set it based on method 3 in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that the system is in maintenance mode.
- **off** indicates that the system is not in maintenance mode.

---

**NOTICE**

Exercise caution when setting this parameter to **on** to avoid data inconsistencies in the cluster.

---

**Default value:** off

## allow\_concurrent\_tuple\_update

**Parameter description:** Specifies whether to allow concurrent update.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that concurrent update is allowed.
- **off** indicates that concurrent update is disallowed.

**Default value:** on

## gtm\_host

**Parameter description:** Specifies the IP address of the primary GTM process. This parameter is visible only to the sysadmin user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string

**Default value:** IP address of the primary GTM

## gtm\_port

**Parameter description:** Specifies the listening port of the primary GTM process. This parameter is visible only to the sysadmin user.

This parameter is a POSTMASTER parameter.

 **NOTE**

This parameter is specified in the configuration file during installation. Do not modify this parameter unless absolutely necessary. Otherwise, database communication will be affected.

**Value range:** an integer ranging from 1 to 65535

**Default value:** specified during installation

## gtm\_host1

**Parameter description:** Specifies the IP address of the standby GTM process. This parameter is visible only to the sysadmin user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string

**Default value:** IP address of the standby GTM

## gtm\_port1

**Parameter description:** Specifies the listening port of the standby GTM process. This parameter is visible only to the sysadmin user.

This parameter is a POSTMASTER parameter.

### NOTE

This parameter is specified in the configuration file during installation. Do not modify this parameter unless absolutely necessary. Otherwise, database communication will be affected.

**Value range:** an integer ranging from 1 to 65535

**Default value:** The value is specified during installation if the standby GTM 1 is deployed. Otherwise, the value is **6665**.

## pgxc\_node\_name

**Parameter description:** Specifies the name of a node.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

When a standby node requests to replicate logs on the primary node, if the **application\_name** parameter is not set, the **pgxc\_node\_name** parameter is used as the name of the streaming replication slot of the standby node on the primary node. The streaming replication slot is named in the following format: Value of this parameter\_IP address of the standby node\_Port number of the standby node. The IP address and port number of the standby node are obtained from the IP address and port number of the standby node specified by the **replconninfo** parameter. The maximum length of a streaming replication slot name is 61 characters. If the length of the concatenated string exceeds 61 characters, the truncated **pgxc\_node\_name** will be used for concatenation to ensure that the length of the streaming replication slot name is less than or equal to 61 characters.

---

### CAUTION

After this parameter is modified, the cluster will fail to be connected. You are advised not to modify this parameter.

---

**Value range:** a string

**Default value:** current node name

## gtm\_backup\_barrier

**Parameter description:** Specifies whether to create a restoration point for the GTM starting point.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that a restoration point will be created for the GTM starting point.
- **off** indicates that a restoration point will not be created for the GTM starting point.

**Default value:** off

## gtm\_conn\_check\_interval

**Parameter description:** Sets the intervals between two consecutive performed checks performed by the CN on the connections between local threads and the primary GTM.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 2147483. The unit is s.

**Default value:** 10s

## transaction\_deferrable

**Parameter description:** Specifies whether to delay the execution of a read-only serial transaction without incurring an execution failure. Assume this parameter is set to **on**. When the server detects that the tuples read by a read-only transaction are being modified by other transactions, it delays the execution of the read-only transaction until the other transactions finish modifying the tuples. This parameter is reserved and does not take effect in this version. Similar to this parameter, the [default\\_transaction\\_deferrable](#) parameter is used to specify whether to allow delayed execution of a transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that the execution of a read-only serial transaction can be delayed.
- **off** indicates that the execution of a read-only serial transaction cannot be delayed.

**Default value:** off

## enable\_show\_any\_tuples

**Parameter description:** This parameter is available only in a read-only transaction and is used for analysis. When this parameter is set to **on** or **true**, all versions of tuples in the table are displayed.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** or **true** indicates that all versions of tuples in the table are displayed.
- **off** or **false** indicates that no versions of tuples in the table are displayed.

**Default value:** off

## gtm\_connect\_timeout

**Parameter description:** Specifies the GTM connection timeout. If the connection time of the GTM exceeds its value, the connection times out and exits.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 2147483647. The unit is s.

**Default value:** 2s

## gtm\_connect\_retries

**Parameter description:** Specifies the number of GTM reconnection attempts.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647

**Default value:** 30

## gtm\_rw\_timeout

**Parameter description:** Specifies the GTM response timeout. If the time spent waiting for GTM responses exceeds its value, the operation times out and exits.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 2147483647. The unit is s.

**Default value:** 1min

## enable\_redistribute

**Parameter description:** Specifies whether unmatched nodes are redistributed.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that unmatched nodes are redistributed.
- **off** indicates that unmatched nodes are not redistributed.

**Default value:** off

## replication\_type

**Parameter description:** Specifies whether the current HA mode is primary/standby/secondary, one primary multiple standbys, or single primary.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

This parameter is used for CM deployment. Do not set it.

**Value range:** 0 to 2

- **0:** indicates that the HA nodes consist of a primary, a standby, and a secondary node.
- **1** Indicates that the one-primary-multiple-standby mode is used, covering all scenarios. This mode is recommended.
- **2** Indicates the single primary mode. In this mode, the standby node cannot be expanded.

**Default value:** 1

## enable\_gtm\_free

**Parameter description:** Specifies whether the GTM-Free mode is enabled. In large concurrency scenarios, the snapshots delivered by the GTM increase in number and size. The network between the GTM and the CN becomes the performance bottleneck. The GTM-Free mode is used to eliminate the bottleneck. In this mode, the CN communicates with DNs instead of the GTM. The CN sends queries to each DN, which locally generates snapshots and xids, ensuring external write consistency but not external read consistency.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

---

 **CAUTION**

When the GTM-Free mode is used, you are advised to set **application\_type** to **perfect\_sharding\_type** so that you can find SQL statements that may cause data inconsistency. Otherwise, the system does not intercept statements that may cause data inconsistency.

---

**Value range:** Boolean

- **on** indicates that the GTM-FREE mode is enabled and the cluster ensures eventual read consistency.
- **off** indicates that the GTM-FREE mode is disabled.

**Default value:** off

## enable\_twophase\_commit

**Parameter description:** Specifies whether to enable distributed two-phase commit in the GTM-Free mode adopted to address the replacement issues of SDS

in the cloud database. This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that distributed two-phase commit is allowed in the GTM-Free mode.
- **off** indicates that distributed two-phase commit is not allowed in the GTM-Free mode.

**Default value:** on

## application\_type

**Parameter description:** valid only when **enable\_gtm\_free** is set to **on**. This parameter specifies the service type of a user. This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#). This parameter cannot be set using **gs\_guc**. Only the following ways are allowed:

1. Use the **gsql** client to perform session-level configuration.
2. When JDBC is used to connect to the database, set the **ApplicationType** parameter for the connection string.

**Value range:** enumerated values

- **not\_perfect\_sharding\_type** indicates a service across nodes. If this value is used, statements across nodes can be executed.
- **perfect\_sharding\_type** indicates a service on a single node. If this value is used and the SQL statement involves multiple nodes, an error is reported. The corresponding SQL statement is recorded in the system log.
  - If this value is used, you can run the **/\*+ multinode \*/ hint** command to allow SQL statements to be executed on multiple nodes. The multinode hint can be added after the select, insert, update, delete, and merge keywords.

## gtm\_host2

**Parameter description:** Specifies the host name or IP address of the standby GTM 2 if the standby GTM 2 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string

**Default value:** the IP address of the standby GTM 2 if the standby GTM 2 is deployed. Otherwise, the value is "".

## gtm\_host3

**Parameter description:** Specifies the host name or IP address of the standby GTM 3 if the standby GTM 3 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string

**Default value:** the IP address of the standby GTM 3 if the standby GTM 3 is deployed. Otherwise, the value is "".

## gtm\_host4

**Parameter description:** Specifies the host name or IP address of the standby GTM 4 if the standby GTM 4 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string

**Default value:** the IP address of the standby GTM 4 if the standby GTM 4 is deployed. Otherwise, the value is "".

## gtm\_host5

**Parameter description:** Specifies the host name or IP address of the standby GTM 5 if the standby GTM 5 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string

**Default value:** the IP address of the standby GTM 5 if the standby GTM 5 is deployed. Otherwise, the value is "".

## gtm\_host6

**Parameter description:** Specifies the host name or IP address of the standby GTM 6 if the standby GTM 6 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string

**Default value:** the IP address of the standby GTM 6 if the standby GTM 6 is deployed. Otherwise, the value is "".

## gtm\_host7

**Parameter description:** Specifies the host name or IP address of the standby GTM 7 if the standby GTM 7 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a string

**Default value:** the IP address of the standby GTM 7 if the standby GTM 7 is deployed. Otherwise, the value is "".

## gtm\_port2

**Parameter description:** Specifies the listening port of the standby GTM 2 if the standby GTM 2 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 65535

**Default value:** The value is specified during installation if the standby GTM 2 is deployed. Otherwise, the value is **6666**.

## gtm\_port3

**Parameter description:** Specifies the listening port of the standby GTM 3 if the standby GTM 3 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 65535

**Default value:** The value is specified during installation if the standby GTM 3 is deployed. Otherwise, the value is **6666**.

## gtm\_port4

**Parameter description:** Specifies the listening port of the standby GTM 4 if the standby GTM 4 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 65535

**Default value:** The value is specified during installation if the standby GTM 4 is deployed. Otherwise, the value is **6666**.

## gtm\_port5

**Parameter description:** Specifies the listening port of the standby GTM 5 if the standby GTM 5 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 65535

**Default value:** The value is specified during installation if the standby GTM 5 is deployed. Otherwise, the value is **6666**.

## gtm\_port6

**Parameter description:** Specifies the listening port of the standby GTM 6 if the standby GTM 6 is deployed. This parameter is visible only to the sysadmin user.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 65535

**Default value:** The value is specified during installation if the standby GTM 6 is deployed. Otherwise, the value is **6666**.

## gtm\_port7

**Parameter description:** Specifies the listening port of the standby GTM 7 if the standby GTM 7 is deployed. This parameter is visible only to the **sysadmin** user.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 65535

**Default value:** The value is specified during installation if the standby GTM 7 is deployed. Otherwise, the value is **6666**.

## enable\_defer\_calculate\_snapshot

**Parameter description:** Specifies the delay in calculating **xmin** and **oldestxmin**. Calculation is triggered only when 1000 transactions are executed or the interval is 1s. If this parameter is set to **on**, the overhead of calculating snapshots can be reduced in heavy-load scenarios, but the progress of **oldestxmin** is slow, affecting tuple recycling. If this parameter is set to **off**, **xmin** and **oldestxmin** can be calculated in real time, but the overhead for calculating snapshots increases.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that snapshots **xmin** and **oldestxmin** are calculated with a delay.
- **off** indicates that snapshots **xmin** and **oldestxmin** are calculated in real time.

**Default value:** on

## 14.3.19 Dual-Cluster Replication Parameters

### enable\_roach\_standby\_cluster

**Parameter description:** Sets the instances of the standby cluster to read-only in dual-cluster mode. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that the read-only mode is enabled for the standby cluster.
- **off** indicates that the read-only mode is disabled for the standby cluster. In this case, the standby cluster can be read and written.

**Default value:** off

## enable\_slot\_log

**Parameter description:** Specifies whether to enable primary/standby synchronization for replication slots. Currently, only archive slots are involved.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that primary/standby synchronization is enabled for replication slots.
- **off** indicates that primary/standby synchronization is disabled for replication slots.

**Default value:** on

## max\_changes\_in\_memory

**Parameter description:** Specifies the maximum number of DML statements cached in memory for a single transaction during logical decoding.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 2147483647

**Default value:** 4096

## max\_cached\_tuplebufs

**Parameter description:** Specifies the upper limit of the total tuple information cached in the memory during logical decoding. You are advised to set this parameter to a value greater than or equal to twice of [max\\_changes\\_in\\_memory](#).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 2147483647

**Default value:** 8192

## logical\_decode\_options\_default

**Parameter description:** Specifies the global default value for unspecified decoding options when logical decoding starts.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

Currently, the following logical decoding options are supported: **parallel-decode-num**, **parallel-queue-size**, **max-txn-in-memory**, **max-reorderbuffer-in-memory**,

and **exclude-users**. For details about the options, see "Application Development Guide > Development Based on JDBC > Example: Logical Replication Code" in *Developer Guide*.

**Value range:** a string of key=value characters separated by commas (,), for example, '**parallel-decode-num=4,parallel-queue-size=128,exclude-users=userA**'. An empty string indicates that the default value hardcoded by the program is used.

**Default value:** ""

---

#### NOTICE

- The SIGHUP parameter does not affect the started logic decoding process. The options specified by this parameter are used as the default settings for subsequent logic decoding startup, and the settings specified in the startup command are preferentially used.
  - The **exclude-users** option is different from the logic decoding startup option. You are not allowed to specify multiple blacklisted users.
- 

## logical\_sender\_timeout

**Parameter description:** Specifies the maximum waiting time for the sender to wait for the receiver to receive logical logs.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is ms.

**Default value:** 30s

## RepOriginId

**Parameter description:** This parameter is a session-level GUC parameter. In bidirectional logical replication, set it to a non-zero value to avoid infinite data replication.

This parameter is a USERSET parameter. Set it based on **Method 3** provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 0

## hadr\_max\_size\_for\_xlog\_receiver

**Parameter description:** Specifies the maximum difference between the OBS logs obtained by instances in the DR cluster and the local replay logs. If the difference is greater than the value of this parameter, the instances stop obtaining OBS logs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Handling suggestion:** The value of this parameter is related to the local disk size. You are advised to set this parameter to 50% of the local disk size.

**Value range:** an integer ranging from 0 to 2147483647. The unit is KB.

**Default value:** 256 GB

## auto\_csn\_barrier

**Parameter description:** Specifies whether the barrier logging function is enabled for the primary cluster for streaming DR.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** off

## stream\_cluster\_run\_mode

**Parameter description:** Specifies whether a CN or DN belongs to the primary or standby cluster in a dual-cluster streaming DR scenario. In a single-cluster scenario, the primary cluster is selected by default.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **cluster\_primary** indicates that the node is in the primary cluster.
- **cluster\_standby** indicates that the node is in the standby cluster.

**Default value:** cluster\_primary

## 14.3.20 Developer Options

### allow\_system\_table\_mods

**Parameter description:** Specifies whether the structure of a system catalog or the name of a system schema can be modified.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the structure of the system catalog or the name of the system schema can be modified.
- **off** indicates that the structure of the system catalog or the name of the system schema cannot be modified.

**Default value:** off

 CAUTION

You are not advised to change the default value of this parameter. If this parameter is set to **on**, system tables may be damaged and the database may fail to be started.

## allow\_create\_sysobject

**Parameter description:** Specifies whether objects such as functions, stored procedures, and synonyms can be created or modified in the system schema. The system schema refers to the schema provided by the database after initialization, excluding the public schema. The OID of the system schema is usually less than 16384.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that initial users and system administrators can create or modify objects such as functions, stored procedures, and synonyms in the system schema. The **sysadmin** user has the permission to create or replace, alter, grant, and revoke system objects by default. For details about whether other users are allowed to create these objects, see the permission requirements of the corresponding schema.
- **off** indicates that all users are not allowed to create or modify objects such as functions, stored procedures, and synonyms in the system schema. The **sysadmin** user does not have the permission to create or replace, alter, grant, or revoke system objects by default.

**Default value:** on

## debug\_assertions

**Parameter description:** Specifies whether to enable various assertion checks. This parameter assists in debugging. If you are experiencing strange problems or crashes, set this parameter to **on** to identify programming defects. To use this parameter, the macro USE\_ASSERT\_CHECKING must be defined (through the configure option **--enable-cassert**) during the GaussDB compilation.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that various assertion checks are enabled.
- **off** indicates that various assertion checks are disabled.

 NOTE

If you compile GaussDB with the assertion check enabled, this parameter is set to **on** by default.

**Default value:** off

## ignore\_checksum\_failure

**Parameter description:** Specifies whether to ignore check failures (but still generate an alarm) and continue reading data. Continuing reading data may result in breakdown, damaged data being transferred or stored, failure of data recovery from remote nodes, or other serious problems. You are not advised to modify the settings.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that data check errors are ignored.
- **off** indicates that data check errors are reported.

**Default value:** off

## ignore\_system\_indexes

**Parameter description:** Specifies whether to ignore system indexes when reading system tables (but still update the indexes when modifying the tables).

This parameter is a BACKEND parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

This parameter is useful for recovering data from tables whose system indexes are damaged.

---

**Value range:** Boolean

- **on** indicates that system indexes are ignored.
- **off** indicates that system indexes are not ignored.

**Default value:** off

## post\_auth\_delay

**Parameter description:** Specifies the delay in the connection to the server after a successful authentication. Developers can attach a debugger to the server startup process.

This parameter is a BACKEND parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147. The unit is second.

**Default value:** 0

 NOTE

This parameter is used only for commissioning and fault locating. To prevent impact on service running, ensure that the default value **0** is used in the production environment. If this parameter is set to a value other than 0, the cluster status may be abnormal due to a long authentication delay.

## pre\_auth\_delay

**Parameter description:** Specifies the period of delaying authentication after the connection to the server is started. Developers can attach a debugger to the authentication procedure.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 60. The unit is s.

**Default value:** 0

 NOTE

This parameter is used only for commissioning and fault locating. To prevent impact on service running, ensure that the default value **0** is used in the production environment. If this parameter is set to a value other than 0, the cluster status may be abnormal due to a long authentication delay.

## trace\_notify

**Parameter description:** Specifies whether to enable the function of generating debugging output for the **LISTEN** and **NOTIFY** commands. The level of [client\\_min\\_messages](#) or [log\\_min\\_messages](#) must be **debug1** or lower so that debugging output can be recorded in the client or server logs, respectively.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

**Default value:** off

## trace\_recovery\_messages

**Parameter description:** Specifies whether to enable logging of recovery-related debugging output. This parameter allows users to overwrite the normal setting of [log\\_min\\_messages](#), but only for specific messages. This is intended for the use in debugging the standby server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values. Valid values include **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, and **log**. For details about the parameter values, see [log\\_min\\_messages](#).

**Default value:** log

 NOTE

- **log** indicates that recovery-related debugging information will not be logged.
- Except the default value **log**, each of the other values indicates that recovery-related debugging information at the specified level will also be logged. Common settings of **log\_min\_messages** enable logs to be unconditionally recorded into server logs.

## trace\_sort

**Parameter description:** Specifies whether to print information about resource usage during sorting operations. This parameter is available only when the macro TRACE\_SORT is defined during the GaussDB compilation. However, TRACE\_SORT is currently defined by default.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

**Default value:** off

## zero\_damaged\_pages

**Parameter description:** Specifies whether to detect a damaged page header that causes GaussDB to report an error, aborting the current transaction.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- Setting this parameter to **on** causes the system to report a warning, zero out the damaged page, and continue processing. This behavior will destroy data, including all the rows on the damaged page. However, it allows you to bypass the error and retrieve rows from any undamaged pages that may be present in the table. Therefore, it is useful for restoring data if corruption has occurred due to a hardware or software error. In most cases, you are advised not to set this parameter to **on** if you want to restore data from damaged pages.
- If this parameter is set to **off**, the system does not fill zeros in damaged pages.

**Default value:** off

## string\_hash\_compatible

**Parameter description:** Specifies whether to use the same method to calculate char-type hash values and varchar- or text-type hash values. Based on the setting of this parameter, you can determine whether a redistribution is required when a distribution column is converted from a char-type data distribution into a varchar- or text-type data distribution.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the same calculation method is used and a redistribution is not required.
- **off** indicates that different calculation methods are used and a redistribution is required.

 **NOTE**

Calculation methods differ in the length of input strings used for calculating hash values. (For a char-type hash value, spaces following a string are not counted as the length. For a text- or varchar-type hash value, the spaces are counted.) The hash value affects the calculation result of queries. To avoid query errors, do not modify this parameter during database running once it is set.

**Default value:** off

## remotetype

**Parameter description:** Specifies the remote connection type.

This parameter is a BACKEND parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values. Valid values are **application**, **coordinator**, **datanode**, **gtm**, **gtmproxy**, **internaltool**, and **gtmtool**.

**Default value:** application

## max\_user\_defined\_exception

**Parameter description:** Specifies the maximum number of exceptions.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer. Currently, only the fixed value **1000** is supported.

**Default value:** 1000

## enable\_compress\_spill

**Parameter description:** Specifies whether to enable the compression function of writing data to disk.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** or **true** indicates that optimization for writing data to disk is enabled.
- **off** or **false** indicates that optimization for writing data to a disk is disabled.

**Default value:** on

## enable\_parallel\_ddl

**Parameter description:** Specifies whether multiple CNs can concurrently perform DDL operations on the same database object.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** DDL operations can be concurrently performed without distributed deadlocks.
- **off:** DDL operations cannot be concurrently performed as distributed deadlocks may occur.

**Default value:** on

## support\_batch\_bind

**Parameter description:** Specifies whether to batch bind and execute PBE statements through interfaces such as JDBC, ODBC, and libpq.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that batch binding and execution are used.
- **off** indicates that batch binding and execution are not used.

**Default value:** on

## numa\_distribute\_mode

**Parameter description:** Specifies the distribution of some shared data and threads among NUMA nodes. This parameter is used to optimize the performance of large-scale ARM servers with multiple NUMA nodes. Generally, you do not need to set this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. The valid values are **none** and **all**.

- **none** indicates that this function is disabled.
- **all** indicates that some shared data and threads are distributed to different NUMA nodes to reduce the number of remote access times and improve performance. Currently, this function applies only to ARM servers with multiple NUMA nodes. All NUMA nodes must be available for database processes. You cannot select only some NUMA nodes.

### NOTE

In the current version, **numa\_distribute\_mode** cannot be set to **all** on the x86 platform.

**Default value:** none

## log\_pagewriter

**Parameter description:** Specifies whether to display the page refresh information of a thread and details about an incremental check point after the incremental check point is enabled. You are not advised to set this parameter to **true** because a large amount of information will be generated.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**Default value:** off

## advance\_xlog\_file\_num

**Parameter description:** Specifies the number of Xlog files that are periodically initialized in advance in the background. This parameter is used to prevent the Xlog file initialization from affecting the performance during transaction submission. However, such a fault may occur only when the system is overloaded. Therefore, you do not need to set this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1000000. The value **0** indicates that initialization is not performed in advance. For example, the value **10** indicates that the background thread periodically initializes 10 Xlog files in advance based on the write location of the current xlog.

**Default value:** 0

## comm\_sender\_buffer\_size

**Parameter description:** Specifies the size of the buffer for each interaction between CNs and DN and between DNs in the stream plan. In some cases, different values affect the stream performance. After the value is reset, the cluster needs to be restarted for the reset to take effect.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 1024. The unit is KB.

**Default value:** 8

## ustore\_attr

**Parameter description:** This parameter is used to control the information statistics of Ustore tables, rollback type, and data verification during the running of key modules (including data, indexes, rollback segments, and playback). This parameter helps R&D engineers locate faults.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. This parameter is set in key-value mode. The mapping between keys and values is as follows: If multiple key-value pairs are used, use

semicolons (;) to separate them. For example, `ustore_attr='ustore_verify_level=FAST;ustore_verify_module=UPAGE:UBTREE:UNDO:REDO'`.

 **NOTE**

When setting `ustore_attr`, do not leave spaces or other characters before and after the equal sign (=) between key and value, for example, `ustore_attr='ustore_verify_level = FAST'`. Otherwise, the parameter is invalid during kernel code verification and the parameter setting fails.

- **ustore\_verify\_level:** Specifies the verification level.  
**Value range:** a string of case-insensitive characters. For details, see the following table.

**Table 14-12** Parameter value meaning of `ustore_verify_level`

Parameter Value	Description
NONE	<b>NONE</b> indicates that the verification function is disabled. You are advised to enable this function to test performance.
FAST	<b>FAST</b> indicates fast verification. A few contents are verified and the impact on performance is minimized.
COMPLETE	<b>COMPLETE</b> indicates complete verification. The verification content is the largest and the performance is greatly affected.

**Default value:** **FAST**

- **ustore\_verify\_module:** Specifies a module that controls verification.  
**Value range:** a string. The value is case-insensitive and can be **UPAGE**, **UBTREE**, **UNDO**, **REDO**, **ROACH**, **ALL**, or **NULL**.  
When more than one of **UPAGE**, **UBTREE**, **UNDO**, **REDO**, and **ROACH** are set, use colons (:) to separate them, for example, `ustore_verify_module=UPAGE:UBTREE:UNDO:REDO`.  
When the **ROACH** module is enabled by users, the value of the `ustore_verify_level` parameter is ignored during the backup of the **ROACH** module. By default, the level of verification is the highest and the performance is greatly affected. Therefore, exercise caution when using the `ustore_verify_module` parameter.

**Table 14-13** Parameter value meaning of `ustore_verify_module`

Parameter Value	Description
UPAGE	Indicates that data page verification is enabled.
UBTREE	Indicates that UB-tree index verification is enabled.

UNDO	Indicates that rollback segment data verification is enabled.
REDO	Indicates that data page verification for the REDO process is enabled.
ROACH	Indicates that data page verification for the ROACH backup is enabled.
ALL	Indicates that data verification is enabled for the UPAGE, UBTREE, UNDO, REDO, and ROACH modules.
NULL	Indicates that data verification for the UPAGE, UBTREE, UNDO, REDO, and ROACH modules is disabled.

**Default value:** UPAGE:UBTREE:UNDO

- **index\_trace\_level:** determines whether to enable index tracing and control the printing level. After this function is enabled, information about index tuples that meet the conditions is printed based on the printing level during index scanning.

**Value range:** a string. The values are described in the following table.

**Default value:** NO

**Table 14-14** Parameter value meaning of index\_trace\_level

Parameter Value	Description
NO	No additional information is printed.
NORMAL	Information about visible index tuples is printed, including: <ul style="list-style-type: none"> <li>• ID and offset of the index page where the current index tuple is located</li> <li>• Current tuple status</li> <li>• TID and partOid corresponding to the current tuple</li> <li>• xmin and xmax information corresponding to the current tuple</li> <li>• Current tuple content (if <b>enable_log_tuple</b> is set to <b>on</b>).</li> </ul>
VISIBILITY	On the basis of <b>NORMAL</b> , the information about the index tuples that do not pass the visibility check is printed and whether the index tuples are visible is marked.
SHOWHIKEY	On the basis of <b>VISIBILITY</b> , the system tries to print the information about the HIKEY tuple on the page.
ALL	Information about all tuples on the scanned index page is printed.

- **enable\_log\_tuple**: specifies whether to print the contents of related tuples when printing log-level prompts for troubleshooting and locating.  
**Value range:** on or off (case-insensitive)  
**Default value:** off
  - **enable\_ustore\_sync\_rollback**: indicates whether to enable synchronous rollback for Ustore tables.  
**Value range:** Boolean  
**Default value:** true
  - **enable\_ustore\_async\_rollback**: indicates whether to enable asynchronous rollback for Ustore tables.  
**Value range:** Boolean  
**Default value:** true
  - **enable\_ustore\_page\_rollback**: indicates whether to enable page rollback for Ustore tables.  
**Value range:** Boolean  
**Default value:** true
  - **enable\_ustore\_partial\_seqscan**: indicates whether to enable partial scan for Ustore tables.  
**Value range:** Boolean  
**Default value:** false
  - **enable\_candidate\_buf\_usage\_count**: indicates whether to enable buffer usage statistics.  
**Value range:** Boolean  
**Default value:** false
  - **ustats\_tracker\_naptime**: specifies the interval for collecting statistics on Ustore tables.  
**Value range:** [1,INT\_MAX/1000]  
**Default value:** 20, in seconds
  - **umax\_search\_length\_for\_prune**: specifies the maximum search depth of the prune operation on the Ustore table.  
**Value range:** [1,INT\_MAX/1000]  
**Default value:** 10 (times)
  - **ustore\_unit\_test**: specifies a test parameter for the white-box test.  
**Value range:** a string  
**Default value:** empty
- Default value:** an empty string

---

 CAUTION

Exercise caution when setting the **ustore\_attr** parameter. You are advised to modify this parameter with the assistance of engineers.

---

## default\_index\_kind

**Parameter description:** Controls the default behavior of creating indexes.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

Value range: an integer. Currently, only the fixed values **0**, **1**, and **2** are supported.

- **0:** The global partition index function is disabled for distributed deployment.
- **1:** A local index is created by default.
- **2:** A global index is created by default.

**Default value:** 2

---

 **CAUTION**

You are advised not to change the default value of this parameter. Otherwise, the index validity may be affected.

---

## 14.3.21 Auditing

### 14.3.21.1 Audit Switch

#### audit\_enabled

**Parameter description:** Specifies whether to enable or disable the audit process. After the audit process is enabled, the auditing information written by the background process can be read from the pipe and written into audit files.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the auditing function is enabled.
- **off** indicates that the auditing function is disabled.

**Default value:** on

#### audit\_directory

**Parameter description:** Specifies the storage directory of audit files. The value can be a path relative to the **data** directory or an absolute path. Only user **sysadmin** can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string.

**Default value:** **pg\_audit** If **om** is used for cluster deployment, audit logs are stored in *\$GAUSSLOG/pg\_audit/Instance name*.

---

**NOTICE**

- You need to set different audit file directories for different CNs or DNs. Otherwise, audit logs will be abnormal.
  - If the value of **audit\_directory** in the configuration file is an invalid path, the audit function cannot be used.
- 

**NOTE**

- Valid path: You have read and write permissions on the path.
- Invalid path: You do not have read or write permissions on the path.

## audit\_data\_format

**Parameter description:** Audits the format of log files. Currently, only the binary format is supported. Only user **sysadmin** can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string.

**Default value:** binary

## audit\_rotation\_interval

**Parameter description:** Specifies the interval of creating an audit log file. If the difference between the current time and the time when the previous audit log file is created is greater than the value of **audit\_rotation\_interval**, a new audit log file will be generated.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 35791394. The unit is min.

**Default value:** 1d

---

**NOTICE**

Do not adjust this parameter unless necessary. Otherwise, **audit\_resource\_policy** may fail to take effect. To control the storage space and time of audit logs, set the [audit\\_resource\\_policy](#), [audit\\_space\\_limit](#), and [audit\\_file\\_remain\\_time](#) parameters.

---

## audit\_rotation\_size

**Parameter description:** Specifies the maximum capacity of an audit log file. If the total number of messages in an audit log exceeds the value of **audit\_rotation\_size**, the server will generate a new audit log file.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1024 to 1048576. The unit is KB.

**Default value:** 10 MB

---

#### NOTICE

- Do not adjust this parameter unless necessary. Otherwise, **audit\_resource\_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit\_resource\_policy**, **audit\_space\_limit**, and **audit\_file\_remain\_time** parameters.
  - If the space occupied by a single record in an audit log file exceeds the value of this parameter, the log file is regarded as an invalid log file.
- 

## audit\_resource\_policy

**Parameter description:** Specifies the policy for determining whether audit logs are preferentially stored by space or time.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that audit logs are preferentially stored by space. A maximum of **audit\_space\_limit** logs can be stored.
- **off** indicates that audit logs are preferentially stored by time. A minimum duration of **audit\_file\_remain\_time** logs must be stored.

**Default value:** on

## audit\_file\_remain\_time

**Parameter description:** Specifies the minimum duration required for recording audit logs. This parameter is valid only when **audit\_resource\_policy** is set to **off**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 730. The unit is day. **0** indicates that the storage duration is not limited.

**Default value:** 90

## audit\_space\_limit

**Parameter description:** Specifies the total disk space occupied by audit files.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1024 KB to 1024 GB. The unit is KB.

**Default value:** 1 GB

---

**NOTICE**

- This parameter takes effect only for a single process instance folder in the **pg\_audit** directory. By default, the total disk space occupied by audit files on each CN or DN is 1 GB.
  - In the multi-audit thread scenario, the minimum disk space occupied by audit files is the product of values of **audit\_thread\_num** and **audit\_rotation\_size**. If the value of this parameter is too small, the disk space occupied by audit files may exceed the value of this parameter.
- 

## audit\_file\_remain\_threshold

**Parameter description:** Specifies the maximum number of audit files in the audit directory.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 100 to 1048576.

**Default value:** 1048576

---

**NOTICE**

- Ensure that this parameter is set to **1048576**. Do not adjust this parameter unless necessary. Otherwise, **audit\_resource\_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit\_resource\_policy**, **audit\_space\_limit**, and **audit\_file\_remain\_time** parameters.
  - In the multi-audit thread scenario, do not adjust this parameter unless necessary. Ensure that the value of this parameter is greater than or equal to the value of **audit\_thread\_num**. Otherwise, the audit function cannot be used and the database is abnormal.
- 

## audit\_thread\_num

**Parameter description:** Specifies the number of audit threads.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 48.

**Default value:** 1

---

**NOTICE**

When **audit\_dml\_state** is enabled and high performance is required, you are advised to increase the value of this parameter to ensure that audit messages can be processed and recorded in a timely manner.

---

## 14.3.21.2 User and Permission Audit

### audit\_login\_logout

**Parameter description:** Specifies whether to audit users' login (including login success and failure) and logout.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 7

- 0 indicates that the function of auditing users' logins and logouts is disabled.
- 1 indicates that only successful user logins are audited.
- 2 indicates that only failed user logins are audited.
- 3 indicates that successful and failed user logins are audited.
- 4 indicates that only user logouts are audited.
- 5 indicates that successful user logouts and logins are audited.
- 6 indicates that failed user logouts and logins are audited.
- 7 indicates that successful user logins, failed user logins, and logouts are audited.

**Default value:** 7

### audit\_database\_process

**Parameter description:** Specifies whether to audit the database startup, stop, switchover, and recovery.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or 1

- 0 indicates that the function of auditing database startup, stop, switchover, and recovery is disabled.
- 1 indicates that the function of auditing database startup, stop, switchover, and recovery is enabled.

**Default value:** 1

### audit\_user\_locked

**Parameter description:** Specifies whether to audit the users' locking and unlocking.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or 1

- 0 indicates that the function of auditing user's locking and unlocking is disabled.
- 1 indicates that the function of auditing user's locking and unlocking is enabled.

**Default value:** 1

## audit\_user\_violation

**Parameter description:** Specifies whether to audit the access violation operations of a user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or 1

- 0 indicates that the function of auditing the access violation operations of a user is disabled.
- 1 indicates that the function of auditing the access violation operations of a user is enabled.

**Default value:** 0

## audit\_grant\_revoke

**Parameter description:** Specifies whether to audit the granting and revoking of user permissions.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or 1

- 0 indicates that the function of auditing the granting and reclaiming of a user's permission is disabled.
- 1 indicates that the function of auditing the granting and reclaiming of a user's permission is enabled.

**Default value:** 1

## full\_audit\_users

**Parameter description:** Specifies the full audit user list. Audit logs are recorded for all auditable operations performed by users in the list.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. Use commas (,) to separate multiple usernames.

**Default value:** empty

## no\_audit\_client

**Parameter description:** Specifies the names and IP addresses of clients that do not need to be audited. The parameter format is *client name@IP address*, which is the same as that of the **client\_conninfo** column in the **pg\_query\_audit** function, for example, *cm\_agent@127.0.0.1* or *gs\_clean@127.0.0.1*.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a character string. Use commas (,) to separate multiple configuration items.

**Default value:** empty

---

**NOTICE**

- If the executed SQL statement meets the configuration requirements of **full\_audit\_users** and **no\_audit\_client**, the **no\_audit\_client** is preferentially configured.
  - Audit logs are generated for communication among tools or nodes in the database server. To save space occupied by audit logs and improve the query performance of audit logs, the low-risk scenarios cannot be audited by configuring the **no\_audit\_client** parameter.
- 

### 14.3.21.3 Operation Auditing

#### audit\_system\_object

**Parameter description:** Specifies whether to audit the CREATE, DROP, and ALTER operations on database objects. Database objects include databases, users, schemas, and tables. You can change the value of this parameter to audit only the operations on required database objects. In the scenario where the leader node is forcibly selected, you are advised to set **audit\_system\_object** to the maximum value and audit all DDL objects.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 536870911

- **0** indicates that the function of auditing the CREATE, DROP, and ALTER operations on database objects can be disabled.
- Other values indicate that the CREATE, DROP, and ALTER operations on a certain or some database objects are audited.

**Value description:**

The value of this parameter is calculated by 29 binary bits. The 29 binary bits represent 29 types of database objects. If the corresponding binary bit is set to **0**, the CREATE, DROP, and ALTER operations on corresponding database objects are not audited. If it is set to **1**, the CREATE, DROP, and ALTER operations are audited. For details about the audit contents represented by these 29 binary bits, see [Table 14-15](#).

When SQL patches are audited and **audit\_dml\_state\_select** are enabled, an SQL patch operation will be audited twice and recorded as DML and DDL operations in the audit log, respectively. If a remote interface is invoked, the DDL logs are generated on the node corresponding to the input parameter, instead of on the node where the statement is issued.

**Default value:** **67121159**, indicating that DDL operations on databases, schemas, users, data sources, node groups, and SQL patches are audited.

**Table 14-15** Meaning of each value for the **audit\_system\_object** parameter

Binary Bit	Description	Value Range
Bit 0	Whether to audit the CREATE, DROP, and ALTER operations on databases.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 1	Whether to audit the CREATE, DROP, and ALTER operations on schemas.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 2	Whether to audit the CREATE, DROP, and ALTER operations on users and user mappings.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 3	Whether to audit the CREATE, DROP, ALTER, and TRUNCATE operations on tables.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are audited.</li> </ul>
Bit 4	Whether to audit the CREATE, DROP, and ALTER operations on indexes.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 5	Whether to audit the CREATE and DROP operations on VIEW and MATVIEW objects.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE and DROP operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE and DROP operations on these objects are audited.</li> </ul>

Binary Bit	Description	Value Range
Bit 6	Whether to audit the CREATE, DROP, and ALTER operations on triggers.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 7	Whether to audit the CREATE, DROP, and ALTER operations on procedures/functions.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 8	Whether to audit the CREATE, DROP, and ALTER operations on tablespaces.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 9	Whether to audit the CREATE, DROP, and ALTER operations on resource pools.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 10	Whether to audit the CREATE, DROP, and ALTER operations on workloads.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 11	Reserved	-
Bit 12	Whether to audit the CREATE, DROP, and ALTER operations on data sources.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 13	Whether to audit the CREATE and DROP operations on node groups.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE and DROP operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE and DROP operations on these objects are audited.</li> </ul>

Binary Bit	Description	Value Range
Bit 14	Whether to audit the CREATE, DROP, and ALTER operations on ROW LEVEL SECURITY objects.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on these objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on these objects are audited.</li> </ul>
Bit 15	Whether to audit the CREATE, DROP, and ALTER operations on types.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on types are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on types are audited.</li> </ul>
Bit 16	Whether to audit the CREATE, DROP, and ALTER operations on text search objects (CONFIGURATION and DICTIONARY).	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on text search objects are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on text search objects are audited.</li> </ul>
Bit 17	Whether to audit the CREATE, DROP, and ALTER operations on directories.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on directories are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on directories are audited.</li> </ul>
Bit 18	Whether to audit the CREATE, DROP, and ALTER operations on synonyms.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on synonyms are not audited.</li> <li>• <b>1</b> indicates that the CREATE, DROP, and ALTER operations on synonyms are audited.</li> </ul>
Bit 19	Whether to audit the CREATE, DROP, and ALTER operations on sequences.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on sequences are not audited.</li> <li>• <b>1</b> indicates that the operations are audited.</li> </ul>
Bit 20	Whether to audit the CREATE, ALTER, and DROP operations on CMKs and CEKs.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, ALTER, and DROP operations on CMKs and CEKs are not audited.</li> <li>• <b>1</b> indicates that the CREATE, ALTER, and DROP operations on CMKs and CEKs are audited.</li> </ul>

Binary Bit	Description	Value Range
Bit 21	Whether to audit the CREATE, DROP, and ALTER operations on packages. (Currently, the operations on packages can be audited only in the centralized deployment scenario.)	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, DROP, and ALTER operations on packages are not audited.</li> <li>• <b>1</b> indicates that the operations are audited.</li> </ul>
Bit 22	Reserved	-
Bit 23	Reserved	-
Bit 24	Whether to audit the ALTER and DROP operations on the <b>gs_global_config</b> objects.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the ALTER and DROP operations on the <b>gs_global_config</b> objects are not audited.</li> <li>• <b>1</b> indicates that the ALTER and DROP operations are audited.</li> </ul>
Bit 25	Reserved	-
Bit 26	Whether to audit the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches are not audited.</li> <li>• <b>1</b> indicates that the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches are audited.</li> </ul>
Bit 27	Reserved	-
Bit 28	Whether to audit the CREATE, ALTER, and DROP operations on database links. Currently, the database link function is not supported.	<ul style="list-style-type: none"> <li>• <b>0</b> indicates that the CREATE, ALTER, and DROP operations on database links are not audited.</li> <li>• <b>1</b> indicates that the CREATE, ALTER, and DROP operations on database links are audited.</li> </ul>

## audit\_dml\_state

**Parameter description:** Specifies whether to audit the INSERT, UPDATE, and DELETE operations on a specific table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer, **0** or **1**

- **0** indicates that the function of auditing the DML operations (except SELECT) is disabled.

- **1** indicates that the function of auditing the DML operations (except SELECT) is enabled.

**Default value:** 0

### **audit\_dml\_state\_select**

**Parameter description:** Specifies whether to audit the SELECT operation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer, **0** or **1**

- **0** indicates that auditing the SELECT operation is disabled.
- **1** indicates that auditing the SELECT operation is enabled.

**Default value:** 0

### **audit\_function\_exec**

**Parameter description:** Specifies whether to record the audit information during the execution of the stored procedures, anonymous blocks, or user-defined functions (excluding system functions).

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** **0** or **1**

- **0** indicates that the function of auditing the procedure or function execution is disabled.
- **1** indicates that the function of auditing the procedure or function execution is enabled.

**Default value:** 0

### **audit\_system\_function\_exec**

**Parameter description:** Specifies whether to record audit logs when system functions in the whitelist are executed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** **0** or **1**

- **0** indicates that the function of auditing the execution of system functions is disabled.
- **1** indicates that the function of auditing system function execution is enabled.

**Default value:** 0

The following table lists the whitelist of system functions that can be audited.

set_working_grand_version_num_manually	set_config	pg_terminate_backend	pg_cancel_backend	pg_cancel_session	pg_cancel_invalid_query
pg_reload_conf	pg_rotate_logfile	pg_terminate_session	pg_terminate_backend	pg_start_backup	pg_stop_backup
pg_create_restore_point	pg_switch_xlog	pg_cbm_get_merged_file	pg_cbm_recycle_file	pg_enable_delay_ddl_recycle	pg_disable_delay_ddl_recycle
pg_cbm_rotate_file	gs_roach_enable_delay_ddl_recycle	gs_roach_disable_delay_ddl_recycle	gs_roach_stop_backup	pg_last_xlog_receive_location	pg_xlog_replay_pause
pg_xlog_replay_resume	gs_roach_switch_xlog	gs_pitr_archive_slot_force_advance	gs_pitr_clean_history_global_barriers	gs_download_obs_file	gs_upload_obs_file
gs_set_obs_file_context	gs_set_obs_delete_location	gs_hadr_dr_switchover	gs_set_obs_delete_location_with_slotname	gs_streaming_dr_in_switchover	pg_advisory_lock
pg_advisory_lock_shared	pg_advisory_unlock	pg_advisory_unlock_shared	pg_advisory_unlock_all	pg_advisory_xact_lock	pg_advisory_xact_lock_shared
pg_try_advisory_lock	pg_try_advisory_lock_shared	pg_try_advisory_xact_lock	pg_try_advisory_xact_lock_shared	gs_get_hadr_key_cn	pg_create_physical_replication_slot_extern
pg_create_logical_replication_slot	pg_drop_replication_slot	pg_logical_slot_peek_changes	pg_logical_slot_get_changes	pg_logical_slot_get_binary_changes	pg_replication_origin_drop
pg_replication_origin_session_reset	local_space_shrink	gs_space_shrink	global_space_shrink	pg_free_remain_segment	gs_fault_inject
sqladvisor.init	sqladvisor.set_weight_params	sqladvisor.set_cost_params	sqladvisor.assign_table_type	gs_repair_file	local_clear_bad_block_info
gs_repair_page	-	-	-	-	-

## audit\_copy\_exec

**Parameter description:** Specifies whether to audit the COPY operation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer, **0** or **1**

- **0** indicates that auditing the COPY operation is disabled.
- **1** indicates that auditing the COPY operation is enabled.

**Default value:** **1**

## audit\_set\_parameter

**Parameter description:** Specifies whether to audit the SET operation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer, **0** or **1**

- **0** indicates that auditing the SET operation is disabled.
- **1** indicates that auditing the SET operation is enabled.

**Default value:** **0**

## audit\_xid\_info

**Parameter description:** Specifies whether to record the transaction ID of the SQL statement in the **detail\_info** column of the audit log.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer, **0** or **1**

- **0** indicates that the function of recording transaction IDs in the audit log is disabled.
- **1** indicates that the function of recording transaction IDs in the audit log is enabled.

**Default value:** **0**

---

### NOTICE

If this function is enabled, the **detail\_info** information in the audit log starts with *xid*. For example:

```
detail_info: xid=14619 , create table t1(id int);
```

If transaction IDs do not exist, *xid* is recorded as **NA** in the audit log.

---

## enableSeparationOfDuty

**Parameter description:** Specifies whether the separation of duties is enabled.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the separation of duties is enabled.
- **off** indicates that the separation of duties is disabled.

**Default value:** off

## enable\_nonsysadmin\_execute\_direct

**Parameter description:** Specifies whether non-system administrator and non-monitor administrator users are allowed to execute the EXECUTE DIRECT ON statement.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that any user is allowed to execute the EXECUTE DIRECT ON statement.
- **off** indicates that only the system administrator and monitor administrator are allowed to execute the EXECUTE DIRECT ON statement.

**Default value:** off

## enable\_access\_server\_directory

**Parameter description:** Specifies whether to allow non-initial users to create, modify, and delete directories.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that non-initial users have the permission to create, modify, and delete directories.
- **off** indicates that non-initial users do not have the permission to create, modify, and delete directories.

**Default value:** off

#### NOTICE

To use the advanced package UTL\_FILE to access files on the server, you must have the permissions on the specified directory.

For security purposes, only the initial user can create, modify, and delete directories by default.

If **enable\_access\_server\_directory** is enabled, users with the **SYSADMIN** permission and users who inherit the **gs\_role\_directory\_create** permission of the built-in role can create directories. A user with the **SYSADMIN** permission, the owner of a directory, a user who is granted with the **DROP** permission for the directory, or a user who inherits the **gs\_role\_directory\_drop** permission of the built-in role can delete the directory. A user with the **SYSADMIN** permission and the owner of a directory object can change the owner of the directory, and the user must be a member of the new owning role.

## 14.3.22 Transaction Monitoring

The automatic rollback transaction can be monitored and its statement problems can be located by setting the transaction timeout warning. In addition, the statements with long execution time can also be monitored.

### transaction\_sync\_naptime

**Parameter description:** For data consistency, when the local transaction's status differs from that in the snapshot of GTM, other transactions will be blocked. You need to wait for a few minutes until the transaction status of the local host is consistent with that of the GTM. The `gs_clean` tool is automatically triggered for cleansing when the waiting period on the CN exceeds that of **transaction\_sync\_naptime**. The tool will shorten the blocking time after it completes the cleansing.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483. The unit is s.

**Default value:** 30s

#### NOTE

If the value of this parameter is set to **0**, `gs_clean` will not be automatically invoked for the cleansing before the blocking arrives the duration. Instead, the `gs_clean` tool is invoked by **gs\_clean\_timeout**. The default value is 5 minutes.

### transaction\_sync\_timeout

**Parameter description:** For data consistency, when the local transaction's status differs from that in the snapshot of GTM, other transactions will be blocked. You need to wait for a few minutes until the transaction status of the local host is consistent with that of the GTM. An exception is reported when the waiting duration on the CN exceeds the value of **transaction\_sync\_timeout**. Roll back the transaction to avoid system blocking due to long time of process response failures (for example, sync lock).

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483. The unit is s.

**Default value:** 10min

 NOTE

- If the value is **0**, no error is reported when the blocking times out or the transaction is rolled back.
- The value of this parameter must be greater than **gs\_clean\_timeout**. Otherwise, unnecessary transaction rollback will probably occur due to a block timeout caused by residual transactions that have not been deleted by **gs\_clean** on the DN.

## 14.3.23 CM Parameters

Modifying CM parameters affects the running mechanism of GaussDB. You are advised to ask GaussDB engineers to do it for you. For details about how to modify the CM parameters, see method 1 in [Table 14-2](#).

You can view CM Agent parameters in the **cm\_agent.conf** file in the CM Agent data directory and CM Server parameters in the **cm\_server.conf** file in the CM Server data directory.

### 14.3.23.1 CM Agent Parameters

#### log\_dir

**Parameter description:** Specifies the directory where CM Agent logs are stored. The value can be an absolute path, or relative to the CM Agent data directory.

**Value range:** a string. Any modification of this parameter takes effect only after CM Agent is restarted. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** "log", indicating that CM Agent logs are generated in the CM Agent data directory.

#### log\_file\_size

**Parameter description:** Specifies the size of a log file. If a log file exceeds the specified size, a new one is created to record log information.

**Value range:** an integer ranging from 0 to 2047. The unit is MB. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 16 MB

#### log\_min\_messages

**Parameter description:** Specifies which message levels are written to the CM Agent log. A higher level covers the messages of all the lower levels. The lower the level is, the fewer messages will be written into the log.

**Value range:** enumerated type. Valid values are **debug5**, **debug1**, **warning**, **error**, **log**, and **fatal**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** warning

## incremental\_build

**Parameter description:** Specifies whether a standby DN is incrementally built. If this function is enabled, the standby DN is incrementally rebuilt. Otherwise, the standby DN is fully rebuilt.

**Value range:** Boolean. The value can be **on** or **off**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** on

## alarm\_component

**Parameter description:** Specifies the location of the alarm component that processes alarms.

**Value range:** a string. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

- If **--alarm-type** in the **gs\_preinstall** script is set to **5**, no third-party component is connected and alarms are written into the **system\_alarm** log. In this case, the value of **alarm\_component** is **/opt/huawei/snas/bin/snas\_cm\_cmd**.
- If **--alarm-type** in the **gs\_preinstall** script is set to **1**, a third-party component is connected. In this case, the value of **alarm\_component** is the absolute path of the executable program of the third-party component.

**Default value:** /opt/huawei/snas/bin/snas\_cm\_cmd

## alarm\_report\_interval

**Parameter description:** Specifies the interval at which an alarm is reported. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Value range:** a non-negative integer. The unit is s.

**Default value:** 1

## alarm\_report\_max\_count

**Parameter description:** Specifies the maximum number of times an alarm is reported. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Value range:** a non-negative integer.

**Default value:** 1

## agent\_report\_interval

**Parameter description:** Specifies the interval at which CM Agent reports the instance status.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1

## agent\_phony\_dead\_check\_interval

**Parameter description:** Specifies the interval at which CM Agent checks whether the CN, DN, or GTM process is suspended.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 10

## agent\_check\_interval

**Parameter description:** Specifies the interval at which the CM Agent queries the status of instances, such as the DNs, CN, and GTM.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 2

## agent\_heartbeat\_timeout

**Parameter description:** Specifies the heartbeat timeout interval for CM Agent to connect to CM Server.

**Value range:** an integer ranging from 2 to  $2^{31} - 1$ . The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 8

## agent\_connect\_timeout

**Parameter description:** Specifies the time to wait before the attempt of CM Agent to connect to CM Server times out.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1

## agent\_connect\_retries

**Parameter description:** Specifies the number of times CM Agent tries to connect to the CM Server.

**Value range:** an integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 15

## agent\_kill\_instance\_timeout

**Parameter description:** Specifies the interval from the time when CM Agent fails to connect to the primary CM Server to the time when CM Agent kills all instances on the node.

**Value range:** an integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 0, indicating that the operation of killing all instances on the node is not initiated.

## enable\_gtm\_phony\_dead\_check

**Parameter description:** Specifies whether to enable the GTM zombie check function.

**Value range:** an integer. The value 1 indicates that the zombie check is enabled, and the value 0 indicates that the zombie check is disabled. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1

## security\_mode

**Parameter description:** Specifies whether CNs and DNPs are started in secure mode. If this parameter is set to **on**, CNs and DNPs are started in secure mode.

**Value range:** Boolean. The value can be **on** or **off**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** off

## upgrade\_from

**Parameter description:** Specifies the internal version number of the cluster before an in-place upgrade. Do not modify the value of this parameter.

**Value range:** a non-negative integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

The recommended value range is [0, *Version number of the installation package*].

**Default value:** 0

## process\_cpu\_affinity

**Parameter description:** Specifies whether to bind a primary DN process to a CPU core before starting the process. If this parameter is set to **0**, core binding will not be performed. If it is set to another value, core binding will be performed, and the number of physical CPU cores is  $2^n$ . Only ARM is supported.

**Value range:** an integer ranging from 0 to 2. The modification of this parameter takes effect only after the cluster and CM Agent are restarted. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 0

## enable\_xc\_maintenance\_mode

**Parameter description:** Specifies whether the **pgxc\_node** system catalog can be modified when the cluster is in read-only mode.

**Value range:** Boolean Any modification of this parameter takes effect only after CM Agent is restarted. For details about how to modify the parameter, see [Table 14-2](#).

- **on** indicates that the **pgxc\_node** system catalog can be modified.
- **off** indicates that the **pgxc\_node** system catalog cannot be modified.

**Default value:** on

## log\_threshold\_check\_interval

**Parameter description:** Specifies the interval for compressing and clearing logs. Typically, this interval is set to 1800s.

**Value range:** an integer ranging from 0 to  $2^{31} - 1$ . The unit is second. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1800

## dilatation\_shard\_count\_for\_disk\_capacity\_alarm

**Parameter description:** Specifies the number of shards to be added in the scale-out scenario. This parameter is used to calculate the threshold for reporting a disk capacity alarm.

### NOTE

The parameter value must be the same as the actual number of shards to be added.

**Value range:** an integer ranging from 0 to  $2^{31} - 1$ . If this parameter is set to **0**, the disk scale-out alarm is not reported. If this parameter is set to a value greater than **0**, the disk scale-out alarm is reported and the threshold is calculated based on the number of shards specified by this parameter. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1

## log\_max\_size

**Parameter description:** Specifies the maximum size of a log file.

**Value range:** an integer ranging from 0 to  $2^{31} - 1$ . The unit is MB. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 10240

## log\_max\_count

**Parameter description:** Specifies the maximum number of logs that can be stored on hard disks.

**Value range:** an integer ranging from 0 to 10000. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 10000

## log\_saved\_days

**Parameter description:** Specifies the number of days for storing logs.

**Value range:** an integer ranging from 0 to 1000. The unit is day. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 90

## enable\_log\_compress

**Parameter description:** Specifies whether to enable log compression.

**Value range:** Boolean The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

- **on** indicates that log compression is enabled.
- **off** indicates that log compression is disabled.

**Default value:** on

## enable\_cn\_auto\_repair

**Parameter description:** Specifies whether to enable automatic CN recovery.

**Value range:** Boolean The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

- **on** indicates that the automatic CN recovery is enabled. That is, after a CN is removed, the agent automatically attempts to recover the CN and add the CN back.
- **off** indicates that automatic CN recovery is disabled.

**Default value:** on

## agent\_backup\_open

**Parameter description:** Specifies whether to enable the DR cluster. After the DR cluster is enabled, the CM runs in DR cluster mode.

**Value range:** an integer, **0** or **1**. Any modification of this parameter takes effect only after CM Agent is restarted. For details about how to modify the parameter, see [Table 14-2](#).

- **0:** disabled.
- **1:** enabled.

**Default value:** **0**

## enable\_e2e\_rto

**Parameter description:** Specifies whether to enable the E2E RTO function. After this function is enabled, the hang-up detection period and network detection timeout time are shortened. The CM can reach the E2E RTO indicator (RTO for a single instance ≤ 10s; RTO for combined faults ≤ 30s).

**Value range:** an integer, **0** or **1**. The value **1** indicates enabled, while the value **0** indicates disabled. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:**

Independent deployment: **1**

## enable\_dcf

**Parameter description:** Specifies the status of the DCF mode.

**Value range:** Boolean The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

- **0:** disabled.
- **1:** enabled.

**Default value:** **off**

## unix\_socket\_directory

**Parameter description:** Specifies the directory location of the Unix socket.

**Value range:** a string. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** **"**

## disaster\_recovery\_type

**Parameter description:** Specifies the type of the DR relationship between primary and standby clusters.

**Value range:** an integer ranging from 0 to 2. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

- 0 indicates that no DR relationship is established.
- 1 indicates that the OBS DR relationship is established.
- 2 indicates that the streaming DR relationship is established.

**Default value:** 0

## environment\_threshold

**Parameter description:** Specifies the thresholds for the physical environment and node status monitored by the agent. If the thresholds are exceeded, logs will be printed. The thresholds include the memory usage threshold, CPU usage threshold, disk usage threshold, instance memory usage threshold, and instance thread pool usage threshold.

**Value range:** a string, in the format of (0, 0, 0, 0). The value range for each number is [0,100]. The unit is %. Value 0 indicates that the detection is disabled. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** (0,0,0,0,0)

### 14.3.23.2 CM Server Parameters

#### log\_dir

**Parameter description:** Specifies the directory where CM Server logs are stored. The value can be an absolute path, or relative to the CM Server data directory.

**Value range:** a string. Any modification of this parameter takes effect only after CM Server is restarted. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** "log", indicating that CM Server logs are generated in the CM Server data directory.

#### log\_file\_size

**Parameter description:** Specifies the size of a log file. If a log file exceeds the specified size, a new one is created to record log information.

**Value range:** an integer ranging from 0 to 2047. The unit is MB. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 16 MB

#### log\_min\_messages

**Parameter description:** Specifies the level of messages to be written to the CM Server log. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

**Value range:** enumerated type. Valid values are **debug5**, **debug1**, **log**, **warning**, **error**, and **fatal**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** warning

## thread\_count

**Parameter description:** Specifies the number of threads in the CM Server thread pool. If the value is greater than the sum of the number of cluster nodes and the number of threads for processing cm\_ctl requests (if the number of cluster nodes is less than 32, one thread is used by default; otherwise, four threads are used), the value that takes effect is the sum of the number of cluster nodes and the number of threads for processing cm\_ctl requests.

**Value range:** an integer ranging from 2 to 1000. Any modification of this parameter takes effect only after CM Server is restarted. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1000

## alarm\_component

**Parameter description:** Specifies the location of the alarm component that processes alarms.

**Value range:** a string. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

- If **--alarm-type** in the **gs\_preinstall** script is set to **5**, no third-party component is connected and alarms are written into the **system\_alarm** log. In this case, the value of **alarm\_component** is **/opt/huawei/snas/bin/snas\_cm\_cmd**.
- If **--alarm-type** in the **gs\_preinstall** script is set to **1**, a third-party component is connected. In this case, the value of **alarm\_component** is the absolute path of the executable program of the third-party component.

**Default value:** /opt/huawei/snas/bin/snas\_cm\_cmd

## instance\_failover\_delay\_timeout

**Parameter description:** Specifies the delay in the CM Server failover when the primary CM Server breakdown is detected.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 0

## instance\_heartbeat\_timeout

**Parameter description:** Specifies the time to wait before the instance heartbeat times out.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 6

### **coordinator\_heartbeat\_timeout**

**Parameter description:** Specifies the heartbeat timeout that triggers the automatic removal of faulty CNs. The setting of this parameter takes effect immediately, and you do not need to restart CM Server. If this parameter is set to 0, faulty CNs are not automatically removed.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 25

### **cmserver\_ha\_connect\_timeout**

**Parameter description:** Specifies the time to wait before the connection between the primary and standby CM Servers times out.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 2

### **cmserver\_ha\_heartbeat\_timeout**

**Parameter description:** Specifies the time to wait before the heartbeat between the primary and standby CM Servers times out.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 6

### **phony\_dead\_effective\_time**

**Parameter description:** Specifies the maximum number of times CN, DN, or GTM processes are detected as zombie. If the number of times a process is detected as zombie is greater than the specified value, the process is considered as a zombie process and will be restarted.

**Value range:** an integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 5

### **enable\_transaction\_read\_only**

**Parameter description:** Specifies whether to enable the automatic threshold detection function of the CM Server disk. After this function is enabled, CM Server

automatically sets the database to read-only when the disk usage is greater than the value of **datastorage\_threshold\_value\_check**.

**Value range:** Boolean values **on**, **off**, **true**, **false**, **yes**, **no**, **1**, and **0** The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** on

## **datastorage\_threshold\_check\_interval**

**Parameter description:** Specifies the interval for checking the disk usage. The system checks the disk usage at the interval specified by the user.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 10

## **datastorage\_threshold\_value\_check**

**Parameter description:** Specifies the usage threshold of a read-only disk in a database. When the disk usage of the data directory exceeds the specified value, the database is automatically set to read-only mode. When adjusting this parameter, you are advised to adjust the **max\_size\_for\_xlog\_retention** parameter of the DN to prevent the cluster read-only threshold from being triggered by backup operations.

**Value range:** an integer ranging from 1 to 99, in percentage. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 85

## **max\_datastorage\_threshold\_check**

**Parameter description:** Specifies the maximum interval for checking the disk usage. After you modify the **enable\_transaction\_read\_only** parameter, the system automatically checks whether the disk usage reaches the threshold at the specified interval.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 43200

## **cmserver\_ha\_status\_interval**

**Parameter description:** Specifies the interval between synchronizations of primary and standby CM Server status.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1

### cmserver\_self\_vote\_timeout

**Parameter description:** Specifies the time to wait before the CM Server self-voting times out.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 6

### alarm\_report\_interval

**Parameter description:** Specifies the interval at which an alarm is reported.

**Value range:** a non-negative integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 3

### alarm\_report\_max\_count

**Parameter description:** Specifies the maximum number of times an alarm is reported.

**Value range:** a non-negative integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1

### enable\_az\_auto\_switchover

**Parameter description:** Specifies whether to enable automatic AZ switchover. If it is set to **1**, CM Server automatically switches over services among AZs. Otherwise, when a DN is faulty, services will not be automatically switched to another AZ even if the current AZ is unavailable. You can run the switchover command to manually switch services to another AZ.

**Value range:** a non-negative integer. The value **0** indicates that automatic AZ switchover is disabled, and the value **1** indicates that automatic AZ switchover is enabled. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1

### instance\_keep\_heartbeat\_timeout

**Parameter description:** The CM Agent periodically checks the instance status and reports the status to the CM Server. If the instance status cannot be detected for a long time and the accumulated number of times exceeds the value of this parameter, the CM Server delivers a command to the CM Agent to restart the instance.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 40

## az\_switchover\_threshold

**Parameter description:** If the failure rate of a DN shard in an AZ (Number of faulty DN shards/Total number of DN shards x 100%) exceeds the specified value, automatic AZ switchover is triggered.

**Value range:** an integer ranging from 0 to 100. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 100

## az\_check\_and\_arbitrate\_interval

**Parameter description:** Specifies the interval for checking the AZ status. If the status of an AZ is abnormal, automatic AZ switchover is triggered.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 2

## az\_connect\_check\_interval

**Parameter description:** Specifies the interval at which the network connection between AZs is checked.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 60

## az\_connect\_check\_delay\_time

**Parameter description:** Specifies the delay between two retries to check the network connection between AZs.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 150

## cmserver\_demote\_delay\_on\_etcd\_fault

**Parameter description:** Specifies the interval at which CM Server switches from the primary state to the standby state due to unhealthy etcd.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 8

## instance\_phony\_dead\_restart\_interval

**Parameter description:** Specifies the interval at which the CM Agent process restarts and kills a zombie CN, DN, or GTM instance. The interval between two consecutive kill operations cannot be less than the value of this parameter. Otherwise, the CM Agent process does not deliver commands.

**Value range:** an integer ranging from 1800 to  $2^{31} - 1$ . The unit is second. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 21600

## cm\_auth\_method

**Parameter description:** Specifies the port authentication mode of the CM. **trust** indicates that port authentication is not configured. **gss** indicates that Kerberos port authentication is used. Note that you can change the value to **gss** only after the Kerberos server and client are successfully installed. Otherwise, the CM cannot communicate properly, affecting the cluster status.

**Value range:** **gss** or **trust**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** trust

## cm\_krb\_server\_keyfile

**Parameter description:** Specifies the location of the key file on the Kerberos server. The value must be an absolute path. The file is usually stored in the `/${GAUSSHOME}/kerberos` directory and ends with `keytab`. The file name is the same as the name of the user who runs the cluster. This parameter is used together with **cm\_auth\_method**. If the **cm\_auth\_method** parameter is changed to **gss**, **cm\_krb\_server\_keyfile** must also be configured as the correct path. Otherwise, the cluster status will be affected.

**Value range:** a string. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** `/${GAUSSHOME}/kerberos/{UserName}.keytab`. The default value cannot take effect and is used only as a prompt.

## cm\_server\_arbitrate\_delay\_base\_time\_out

**Parameter description:** Specifies the basic delay duration for CM Server arbitration. If the primary CM Server is disconnected, the arbitration starts to be timed. If the disconnection duration exceeds the arbitration delay duration, a new primary CM Server will be selected. The arbitration delay duration is determined by the basic delay duration, the node index (server ID), and the incremental delay

duration. The formula is as follows: Arbitration delay duration = Basic delay duration + Node index x Incremental delay duration

**Value range:** an integer. The unit is s. The index should be larger than 0. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 10

## cm\_server\_arbitrate\_delay\_incremental\_time\_out

**Parameter description:** Specifies the incremental delay duration for CM Server arbitration. If the primary CM Server is disconnected, the arbitration starts to be timed. If the disconnection duration exceeds the arbitration delay duration, a new primary CM Server will be selected. The arbitration delay duration is determined by the basic delay duration, the node index (server ID), and the incremental delay duration. The formula is as follows: Arbitration delay duration = Basic delay duration + Node index x Incremental delay duration

**Value range:** an integer. The unit is s. The index should be larger than 0. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 3

## force\_promote

**Parameter description:** Specifies whether CM Server enables the forcible startup logic (that is, when the cluster status is unknown, ensure that the basic functions of the cluster are available at the cost of partial data loss). The value **0** indicates that forcible startup is disabled, and the value **1** indicates that forcible startup is enabled. This parameter applies to CNs and DNs.

**Value range:** **0** or **1**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 0

## switch\_rto

**Parameter description:** Specifies the delay for the forcible switchover of CM Server. When **force\_promote** is set to **1** and a shard in the cluster does not have primary CM Server, the system starts timing. After the delay, the forcible startup logic starts to be executed.

**Value range:** an integer ranging from 0 to 2147483647. The unit is second. The minimum value that takes effect is 60. If this parameter is set to a value less than 60, 60s is used. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 600

## enable\_finishredo\_retrieve

**Parameter description:** Specifies whether to retrieve data of Xlogs that have been cut off by redo after a forcible CM Server switchover. If this parameter is set to **on**, data is automatically retrieved after a forcible switchover.

**Value range:** Boolean. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

- **off:** disabled.
- **on:** enabled.

**Default value:** off

## backup\_open

**Parameter description:** Specifies whether to enable the DR cluster. After the DR cluster is enabled, the CM runs in DR cluster mode.

**Value range:** an integer, **0** or **1**. Any modification of this parameter takes effect only after CM Server is restarted. This parameter cannot be enabled for non-DR clusters. For details about how to modify the parameter, see [Table 14-2](#).

- **0:** disabled.
- **1:** enabled.

**Default value:** 0

## enable\_e2e\_rto

**Parameter description:** Specifies whether to enable the E2E RTO function. After this function is enabled, the hang-up detection period and network detection timeout time are shortened. The CM can reach the E2E RTO indicator (RTO for a single instance  $\leq 10s$ ; RTO for combined faults  $\leq 30s$ ).

**Value range:** **0** or **1**. The value **1** indicates enabled, while the value **0** indicates disabled. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:**

Independent deployment: **1**

## cluster\_starting\_aribt\_delay

**Parameter description:** Specifies the time that CM Server waits for the static primary DN to be promoted to primary during cluster startup.

**Value range:** an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 180

## enable\_dcf

**Parameter description:** Specifies the status of the DCF mode.

**Value range:** Boolean The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

- 0: disabled.
- 1: enabled.

**Default value:** off

## ddb\_type

**Parameter description:** Specifies whether to switch between ETCD and DCC modes.

**Value range:** an integer. 0: ETCD; 1: DCC. Any modification of this parameter takes effect only after CM Server is restarted. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 0

## enable\_ssl

**Parameter description:** Specifies whether to enable SSL.

**Value range:** Boolean After this function is enabled, the SSL certificate is used to encrypt communication. Any modification of this parameter takes effect only after CM Server is restarted. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:**

- on indicates that SSL is enabled.
- off indicates that SSL is disabled.
- **Default value:** off

---

### NOTICE

To ensure security, you are not advised to disable it. After this function is disabled, the CM does not use encrypted communication and all information is transmitted in plaintext, which may bring security risks such as eavesdropping, tampering, and spoofing.

---

## ssl\_cert\_expire\_alert\_threshold

**Parameter description:** Specifies the SSL certificate expiration alerting time.

**Value range:** an integer. The unit is day. If the certificate expiration time is less than the value of this parameter, an alarm indicating that the certificate is about to expire is reported. Any modification of this parameter takes effect only after CM Server is restarted. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 90

## ssl\_cert\_expire\_check\_interval

**Parameter description:** Specifies the period for checking whether the SSL certificate expires.

**Value range:** an integer. The unit is s. Any modification of this parameter takes effect only after CM Server is restarted. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 86400

## ddb\_log\_level

**Parameter description:** Sets the DDB log level.

To disable the log function, set this parameter to **NONE**, which cannot be used together with the following log levels:

To enable the log function, set this parameter to one or a combination of the following log levels: **RUN\_ERR|RUN\_WAR|RUN\_INF|DEBUG\_ERR|DEBUG\_WAR|DEBUG\_INF|TRACE|PROFILE|OPER**. If two or more log levels are used together, separate them with vertical bars (|). The log level cannot be set to an empty string.

**Value range:** a string containing one or a combination of the following log levels: **RUN\_ERR|RUN\_WAR|RUN\_INF|DEBUG\_ERR|DEBUG\_WAR|DEBUG\_INF|TRACE|PROFILE|OPER**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** RUN\_ERR|RUN\_WAR|DEBUG\_ERR|OPER|RUN\_INF|PROFILE

## ddb\_log\_backup\_file\_count

**Parameter description:** Specifies the maximum number of log files that can be saved.

**Value range:** an integer ranging from 1 to 100. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 10

## ddb\_max\_log\_file\_size

**Parameter description:** Specifies the maximum number of bytes in a log.

**Value range:** a string, in the range [1 MB, 1000 MB]. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 10 MB

## ddb\_log\_suppress\_enable

**Parameter description:** Specifies whether to enable the log suppression function.

**Value range:** an integer. **0:** disabled; **1:** enabled. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 1

## ddb\_election\_timeout

**Parameter description:** Specifies the DCC election timeout period.

**Value range:** an integer, in the range [1,600], in seconds. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 3

## delay\_arbitrate\_timeout

**Parameter description:** Specifies the waiting time for a node in the same AZ as the primary DN to be promoted to primary after redo replay.

**Value range:** an integer, in the range [0,21474836] (unit: second). The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see [Table 14-2](#).

**Default value:** 0

## install\_type

**Parameter description:** Specifies the settings related to the DR cluster to distinguish the cluster type.

**Value range:** an integer ranging from 0 to 2. Any modification of this parameter takes effect only after CM Server is restarted. This parameter cannot be enabled for non-DR clusters. For details about how to modify the parameter, see [Table 14-1](#).

**Default value:** 0

- **0** indicates the cluster for which no DR relationship is established.
- **1** indicates a Dorado-based cluster.
- **2** indicates a streaming-based cluster.

## enable\_synclist\_single\_inst

**Parameter description:** Specifies whether to reduce copies to one primary and zero standby.

**Value range:** Boolean. If this function is enabled, the copies are reduced to one primary and zero standby. If this parameter is incorrectly set, the default value is used. The modification of this parameter takes effect after reloading. For details about how to modify parameters, see the **Set cm parameter** table in section "Unified Database Management Tool > cm\_ctl Tool Introduction" in *Tool Reference*.

- **off:** indicates that this function is disabled.

- **on**: indicates that this function is enabled.

**Default value:** off

## 14.3.24 GTM Parameters

GTM parameters can be set in the **gtm.conf** file or using **gs\_guc**.

### nodename

**Parameter description:** Specifies the name of the primary or standby GTM.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, which complies with the identifier naming convention

**Default value:** NULL

### port

**Parameter description:** Specifies the host port number listened by the primary or standby GTM.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The recommended value range is 1024 to 65535.

**Default value:** 6666

### log\_file

**Parameter description:** Specifies a log file name.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, which complies with the identifier naming convention

**Default value:** `gtm-%Y-%m-%d_%H%M%S.log`

### active\_host

**Parameter description:** Specifies the IP address of a target GTM. For the primary GTM, it is the IP address of the standby GTM; for the standby GTM, it is the IP address of the primary GTM.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, which complies with the identifier naming convention

**Default value:** NULL

## local\_host

**Parameter description:** Specifies the HA local address. Set this parameter based on the cluster configuration file. You do not need to manually set this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, which complies with the identifier naming convention.

**Default value:** NULL

## active\_port

**Parameter description:** Specifies the port number of the target GTM server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The recommended value range is 1024 to 65535.

 **NOTE**

This parameter is specified in the configuration file during installation. Do not modify this parameter unless absolutely necessary. Otherwise, database communication will be affected.

**Default value:** 0

## local\_port

**Parameter description:** Specifies the local port for HA.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The recommended value range is 1024 to 65535.

 **NOTE**

This parameter is specified in the configuration file during installation. Do not modify this parameter unless absolutely necessary. Otherwise, database communication will be affected.

**Default value:** 0

## standby\_connection\_timeout

**Parameter description:** Specifies the timeout interval between the primary and standby GTMs. A larger value enhances the fault tolerance capability of the network between the primary and standby GTMs, but increases the duration for reporting disconnection between them.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 5 to 2147483647. The unit is s.

**Default value:** 5

## keepalives\_count

**Parameter description:** Specifies the number of keepalived signals that can be waited before the GTM server is disconnected from the client if the OS supports the **TCP\_KEEPCNT** socket parameter. This parameter takes effect only on the standby GTM.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 0, indicating that the connection is immediately broken if no keepalived signal from the client is received by the GTM.

## keepalives\_idle

**Parameter description:** Specifies the interval for sending keepalived signals.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is s.

**Default value:** 0

## keepalives\_interval

**Parameter description:** Specifies the response time before retransmission on an OS that supports the **TCP\_KEEPINTVL** socket option.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647. The unit is s.

**Default value:** 0

## synchronous\_backup

**Parameter description:** Specifies whether to enable synchronization for backing up data to the standby GTM.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** **off**, **on**, or **auto**

- **on:** Synchronization is enabled.
- **off:** Synchronization is disabled.
- **auto:** Automatic synchronization is enabled.

**Default value:** **auto**

## query\_memory\_limit

**Parameter description:** Specifies the limit of memory available for queries. This parameter applies only to the default resource group. For other resource groups, the memory available for queries is not limited.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a floating point number ranging from **0.0** to **1.0**

**Default value:** **0.25**

## wlm\_max\_mem

**Parameter description:** Specifies the maximum memory for GTM execution.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 512 MB to 2147483647 MB.

**Default value:** **2048**

## config\_file

**Parameter description:** Specifies the name of a GTM configuration file. Only the sysadmin user can access this parameter.

**Value range:** a string Set it based on instructions provided in [Table 14-1](#).

**Default value:** **gtm.conf**

## data\_dir

**Parameter description:** Specifies the GTM data file directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** **NULL**

## listen\_addresses

**Parameter description:** Specifies the TCP/IP address of the client for a server to listen on.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:**

- Host name or IP address. Multiple values are separated with commas (,).
- An asterisk (\*), indicating all IP addresses.
- If the parameter is not specified, the server does not listen on any IP address. In this case, only Unix domain sockets can be used for database connections.

**Default value:** \*

## log\_directory

**Parameter description:** Specifies the directory for storing log files when **logging\_collector** is set to **on**. The value can be an absolute path, or relative to the data directory.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

### NOTICE

- If the value of **log\_directory** in the configuration file is an invalid path (that is, the user does not have the permission to read or write this path), the cluster cannot be restarted.
- If the value of **log\_directory** is changed to a valid path (that is, the user has the permission to read and write this path), logs are generated in the new path. If the specified path is invalid, log files are generated in the last valid path and the database running is not affected. The invalid value is still written into the configuration file.

---

**Value range:** a string

**Default value:** **gtm\_log**, indicating that server logs will be generated in the **gtm\_log/** directory under the data directory.

## log\_min\_messages

**Parameter description:** Specifies which level of messages will be written into server logs. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

---

### NOTICE

If the values of **client\_min\_messages** and **log\_min\_messages** are the same, they indicate different levels.

---

**Valid values:** enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details, see [Table 14-6](#).

**Default value:** **warning**

## alarm\_component

**Parameter description:** Certain alarms are suppressed during alarm reporting. That is, the same alarm will not be repeatedly reported by an instance within the period specified by **alarm\_report\_interval**. Its default value is **10s**. In this case, the parameter specifies the location of the alarm component that is used to process alarm information.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

- If **--alarm-type** in the **gs\_preinstall** script is set to **5**, no third-party component is connected and alarms are written into the **system\_alarm** log. In this case, the value of **alarm\_component** is **/opt/huawei/snas/bin/snas\_cm\_cmd**.
- If **--alarm-type** in the **gs\_preinstall** script is set to **1**, a third-party component is connected. In this case, the value of **alarm\_component** is the absolute path of the executable program of the third-party component.

**Default value:** **/opt/huawei/snas/bin/snas\_cm\_cmd**

## alarm\_report\_interval

**Parameter description:** Specifies the interval at which an alarm is reported.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a non-negative integer (unit: s)

**Default value:** **10**

## standby\_only

**Parameter description:** Specifies whether to forcibly synchronize information to standby nodes. In one primary+multiple standbys mode, information is only forcibly synchronized to the ETCD.

**Value range:** **0** or **1**. Set this parameter based on instructions provided in [Table 14-1](#).

- **0**: Information is not forcibly synchronized to standby nodes.
- **1**: Information is forcibly synchronized to standby nodes.

**Default value:** **0**

## gtm\_max\_trans

**Parameter description:** Specifies the maximum number of connections accepted by the GTM. You are not advised to change the value. If you have to, set this parameter to a value no less than the maximum number of connections plus 100.

**Value range:** an integer ranging from 256 to 200000. Set it based on instructions provided in [Table 14-1](#).

**Default value:** **8192**

## enable\_connect\_control

**Parameter description:** Specifies whether the GTM verifies that a connection IP address is within the cluster.

**Value range:** Boolean. Set it based on instructions provided in [Table 14-1](#).

- **true**: The GTM checks whether a connection IP address is within the cluster. If it is not, the access is rejected.
- **false**: The GTM does not check whether a connection IP address is within the cluster.

**Default value:** true

## gtm\_authentication\_type

**Parameter description:** Specifies the port authentication mode of the GTM. **trust** indicates that port authentication is not configured. **gss** indicates that Kerberos port authentication is used. Note that you can change the value to **gss** only after the Kerberos server and client are successfully installed. Otherwise, the GTM cannot communicate properly, affecting the cluster status.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** gss or trust.

**Default value:** trust

## gtm\_krb\_server\_keyfile

**Parameter description:** Specifies the location of the key file on the Kerberos server. The value must be an absolute path. The file is usually stored in the  $\$ \{GAUSSHOME\}/\text{kerberos}$  directory and ends with keytab. The file name is the same as the name of the user who runs the cluster. This parameter is used together with **gtm\_authentication\_type**. If **gtm\_authentication\_type** is changed to **gss**, **gtm\_krb\_server\_keyfile** must be configured as the correct path. Otherwise, the cluster status will be affected.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** "

## gtm\_option

**Parameter description:** Specifies the GTM mode, which must be set to the same value on all GTMs, CNs, and DNS. There are three GTM modes: GTM, GTM-Lite, and GTM-FREE. For details, see "GTM Mode" in *Feature Guide*. The GTM and GTM-Lite modes take effect only when the **enable\_gtm\_free** parameter is set to **off**. The current version does not support switching between different GTM modes for installed clusters.

**Value range:** an integer ranging from 0 to 2. The value **0** indicates the GTM mode, the value **1** indicates the GTM-Lite mode, and the value **2** indicates the GTM-FREE mode. Set it based on instructions provided in [Table 14-1](#).

**Default value:** 1

## csn\_sync\_interval

**Parameter description:** Specifies the interval for synchronizing CSN between the primary and standby GTMs.

**Value range:** an integer ranging from 1 to 2147483647 . The unit is s. Set it based on instructions provided in [Table 14-1](#).

**Default value:** 1

## restore\_duration

**Parameter description:** Specifies the pre-allocation interval of XID or CSN on the GTM.

**Value range:** an integer ranging from 1000000 to 2147483647. Set it based on instructions provided in [Table 14-1](#).

**Default value:** 1000000

## gtm\_enable\_threadpool

**Parameter description:** Specifies whether to enable the GTM thread pool function. The setting takes effect only after the GTM is restarted.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**Default value:** true

## gtm\_num\_threads

**Parameter description:** Specifies the number of work threads in the thread pool when the thread pool function **gtm\_enable\_threadpool** is enabled.

The value is related to the size of **gtm\_max\_trans** and cannot exceed the result of (Value of **gtm\_max\_trans** - 1 - Number of auxiliary threads). The number of auxiliary threads is 2 in the current version.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 16384

**Default value:** 1024

## 14.3.25 Upgrade Parameters

### IsInplaceUpgrade

**Parameter description:** Specifies whether an upgrade is ongoing. This parameter is an upgrade parameter and cannot be modified. Only the sysadmin user can access the parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates an upgrade is ongoing.
- **off** indicates no upgrade is ongoing.

**Default value:** off

## inplace\_upgrade\_next\_system\_object\_oids

**Parameter description:** Specifies the OID of a new system object during the in-place upgrade. This parameter is used for upgrade and cannot be modified by users.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** empty

## upgrade\_mode

**Parameter description:** Specifies the upgrade mode. This parameter is used for upgrade. You are advised not to modify it.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Value range:** an integer ranging from 0 to 2147483647

- **0:** indicates that the local upgrade and the minor version gray upgrade are not in progress.
- **1:** indicates that the upgrade is in progress. The upgrade command is executed and takes effect after the check is completed.
- **2:** indicates that the major version gray upgrade is in progress. The upgrade command is executed and takes effect after the check is completed.

**Default value:** 0

### NOTE

Execute the precommand on new packages, switch to the cluster user, and use the **source** command to invalidate environment variables. Run the **gs\_upgradectl -t chose-strategy** command to check whether the upgrade is a major version upgrade or minor version upgrade.

If "Upgrade strategy: large-binary-upgrade" is returned, the major version is upgraded.

If "Upgrade strategy: small-binary-upgrade" is returned, the minor version is upgraded.

## 14.3.26 Miscellaneous Parameters

### enable\_default\_ustore\_table

**Parameter description:** Specifies whether to enable the Ustore storage engine by default. If this parameter is set to **on**, all created tables are Ustore tables.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#). Note that the **track\_counts** and **track\_activities** parameters must be enabled when the Ustore table is used. Otherwise, space expansion may occur.

**Value range:** [off, on]

**Default value:** off

## enable\_ustore

**Parameter description:** Specifies whether to enable the Ustore storage engine. If this parameter is set to **on**, Ustore tables can be created. Note that the **track\_counts** and **track\_activities** parameters must be enabled when the Ustore table is used. Otherwise, space expansion may occur.

This is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** [off, on]

**Default value:** on

## reserve\_space\_for\_nullable\_atts

**Parameter description:** Specifies whether to reserve space for the nullable attribute of an Ustore table. If this parameter is set to **on**, space is reserved for the nullable attribute of the Ustore table by default.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** [off, on]

**Default value:** on

## server\_version

**Parameter description:** Specifies the server version number (string).

This parameter is a fixed INTERNAL parameter and cannot be modified. This parameter is inherited from the PostgreSQL kernel, indicating that the current database kernel is compatible with PostgreSQL's **server\_version**. This parameter is reserved to ensure the ecosystem compatibility of the northbound external tool APIs (which will be queried when the tool is connected). This parameter is not recommended. You can use the **opengauss\_version()** function to obtain the kernel version.

**Value range:** a string.

**Default value:** 9.2.4

## server\_version\_num

**Parameter description:** Specifies the server version number (integer).

This parameter is a fixed INTERNAL parameter and cannot be modified. This parameter is inherited from the PostgreSQL kernel, indicating that the current database kernel is compatible with PostgreSQL's **server\_version\_num**. This parameter is reserved to ensure the ecosystem compatibility of the northbound external tool APIs (which will be queried when the tool is connected).

**Value range:** an integer

**Default value:** 90204

## block\_size

**Parameter description:** Specifies the block size of the current database.

This parameter is a fixed INTERNAL parameter and cannot be modified. This parameter is inherited from the PostgreSQL kernel, indicating that the current database kernel is compatible with PostgreSQL's **server\_version\_num**. This parameter is reserved to ensure the ecosystem compatibility of the northbound external tool APIs.

**Default value:** 8192

## segment\_size

**Parameter description:** Specifies the segment file size of the current database.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Default value:** 1 GB

## max\_index\_keys

**Parameter description:** Specifies the maximum number of index keys supported by the current database.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Default value:** 32

## integer\_datetimes

**Parameter description:** Specifies whether the date and time are in the 64-bit integer format.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Value range:** Boolean

- **on** indicates that the 64-bit integer format is used.
- **off** indicates that the 64-bit integer format is not used.

**Default value:** on

## enable\_cluster\_resize

**Parameter description:** If an SQL statement involves tables belonging to different groups, you can enable this parameter to push the execution plan of the statement to improve performance.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the execution plan of the statement can be pushed.

- **off** indicates that the execution plan of the statement cannot be pushed.

**Default value:** off

 **NOTE**

This parameter is used for internal O&M. Do not set it to **on** unless absolutely necessary.

## lc\_collate

**Parameter description:** Specifies the locale in which sorting of textual data is done.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Default value:** Determined by the configuration set during the cluster installation and deployment.

## lc\_ctype

**Parameter description:** Specifies the locale that determines character classifications. For example, it specifies what a letter and its upper-case equivalent are.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Default value:** Determined by the configuration set during the cluster installation and deployment.

## max\_identifier\_length

**Parameter description:** Specifies the maximum identifier length.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Value range:** an integer

**Default value:** 63

## server\_encoding

**Parameter description:** Specifies the database encoding (character set).

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

**Default value:** Determined when the database is created.

## enable\_upgrade\_merge\_lock\_mode

**Parameter description:** If this parameter is set to **on**, the delta merge operation internally increases the lock level, and errors can be prevented when update and delete operations are performed at the same time.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- If this parameter is set to **on**, the delta merge operation internally increases the lock level. In this way, when the **DELTAMERGE** operation is concurrently performed with the **UPDATE** or **DELETE** operation, one operation can be performed only after the previous one is complete.
- If this parameter is set to **off** and the **DELTAMERGE** operation is concurrently performed with the **UPDATE** or **DELETE** operation to the data in a row in the delta table of the table, errors will be reported during the later operation, and the operation will stop.

**Default value:** off

## transparent\_encrypt\_kms\_region

**Parameter description:** This parameter is discarded. It stores the deployment region of the entire cluster. The content must contain only the characters specified in RFC3986, and the maximum length is 2047 bytes.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string.

**Default value:** empty.

## datanode\_heartbeat\_interval

**Parameter description:** Specifies the interval at which heartbeat messages are sent between heartbeat threads. You are advised to set this parameter to a value no more than **wal\_receiver\_timeout/2**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1000 to 60000. The unit is ms.

**Default value:** 1s

## dfs\_partition\_directory\_length

**Parameter description:** Specifies the maximum directory name length for the partition directory of a table partitioned by VALUE in the HDFS.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 92 to 7999

**Default value:** 512

## max\_concurrent\_autonomous\_transactions

**Parameter description:** Specifies the maximum number of autonomous transaction connections, that is, the maximum number of concurrent autonomous transactions executed at the same time. If this parameter is set to **0**, autonomous transactions cannot be executed.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0–10000. The theoretical maximum value is **10000**, and the actual maximum value is a dynamic value. The calculation formula is  $262143 - \text{job\_queue\_processes} - \text{autovacuum\_max\_workers} - \text{max\_inner\_tool\_connections} - \text{max\_connections} - \text{AUXILIARY\_BACKENDS} - \text{AV\_LAUNCHER\_PROCS}$ . The values of [job\\\_queue\\\_processes](#), [autovacuum\\\_max\\\_workers](#), [max\\\_inner\\\_tool\\\_connections](#), and [max\\\_connections](#) depend on the settings of the corresponding GUC parameters. **AUXILIARY\\_BACKENDS** indicates the number of reserved auxiliary threads, which is fixed to **20**. **AV\\_LAUNCHER\\_PROCS** indicates the number of reserved launcher threads for autovacuum, which is fixed to **2**.

**Default value:**

Independent deployment: **80** (60-core CPU/480 GB memory); **40** (32-core CPU/256 GB memory); **20** (16-core CPU/128 GB memory); **10** (8-core CPU/64 GB memory, 4-core CPU/32 GB memory, 4-core CPU/16 GB memory)

**Suggestion:** Set this parameter based on actual service requirements and hardware configurations. It is recommended that this parameter be set to a value less than or equal to 1/10 of **max\\_connections**. If you only increase the value of this parameter but do not adjust the memory parameters in the same proportion, the memory may be insufficient and the error message "memory is temporarily unavailable" is displayed when the service load is heavy.

 **NOTE**

If the value range of this parameter is changed during the upgrade and the value is changed before the commit operation, you need to change the value range to the value allowed before the upgrade if you roll back the upgrade. Otherwise, the database may fail to be started.

## mot\_config\_file

This parameter is unavailable in a distributed system.

## enable\_gpi\_auto\_update

**Parameter description:** Determines whether global indexes are updated by default in partition DDL commands.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#). This parameter must be set to the same value on CNs and DNPs. Otherwise, the global index function may be abnormal.

**Value range:** Boolean

- The value **on** indicates that global indexes are updated regardless of whether the partition DDL commands contain the UPDATE GLOBAL INDEX clause.
- The value **off** indicates that global indexes are not updated unless the partition DDL commands contain the UPDATE GLOBAL INDEX clause.

**Default value:** off

## change\_cluster\_mode

**Parameter description:** Specifies whether the cluster is in the mode switchover process. Mode switchover refers to: A cluster with one primary and two standby nodes is switched to a cluster with one primary node, one standby node, and one log; a cluster with one primary node, one standby node, and one log is switched to a cluster with one primary and two standby nodes; standby DNs and log DNs in a cluster with one primary node, one standby node, and one log are switched to each other.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

- The value **0** indicates that the cluster mode is not being switched.
- The value **1** indicates that the cluster mode is being switched.
- Other values are meaningless and have the same effect as the value **0**.

**Default value:** 0

---

 **CAUTION**

This parameter specifies whether the read function of the standby node in a distributed system is available. If this parameter is set to **1**, the read function of the standby node in a distributed system cannot be used. Exercise caution when setting this parameter.

---

## 14.3.27 Wait Event

### enable\_instr\_track\_wait

**Parameter description:** Specifies whether to enable real-time collection of wait event information.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the function of collecting wait event information is enabled.
- **off** indicates that the function of collecting wait event information is disabled.

**Default value:** on

## 14.3.28 Query

### instr\_unique\_sql\_count

**Parameter description:** Specifies the maximum number of unique SQL records to be collected. The value **0** indicates that the function of collecting unique SQL information is disabled.

If the value is changed from a larger one to a smaller one, unique SQL statistics will be reset and re-collected (the standby node does not support this function). There is no impact if the value is changed from a smaller one to a larger one.

When the number of unique SQL records generated in the system (to view the statistics, query **dbperf.statement** or **dbperf.summary\_statement**) is greater than the value of **instr\_unique\_sql\_count**, the extra unique SQL records are not collected.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU/256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 3% by enabling or disabling this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 200000

## instr\_unique\_sql\_track\_type

**Parameter description:** Specifies which SQL statements are recorded in Unique SQL.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values

- **top:** Only top-level SQL statements are recorded.
- **all:** All SQL statements are recorded.

**Default value:** top

## unique\_sql\_retention\_time

**Parameter description:** Specifies the interval for cleaning the unique SQL hash table. The default value is 30 minutes.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 3650. The unit is minute.

**Default value:** 30min

## enable\_instr\_rt\_percentile

**Parameter description:** Specifies whether to enable the function of calculating the response time of 80% and 95% SQL statements in the system.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the function of calculating the response time of 80% and 95% SQL statements is enabled.
- **off** indicates that the function of calculating the response time of 80% and 95% SQL statements is disabled.

**Default value:** on

## percentile

**Parameter description:** Specifies the percentage of SQL statements whose response time is to be calculated by the background calculation thread.

This parameter is an INTERNAL parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** "80,95"

## instr\_rt\_percentile\_interval

**Parameter description:** Specifies the interval at which the background calculation thread calculates the SQL response time.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 3600. The unit is s.

**Default value:** 10s

## enable\_instr\_cpu\_timer

**Parameter description:** Specifies whether to capture the CPU time consumed during SQL statement execution.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU/256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 3.5% by enabling or disabling this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the CPU time consumed during SQL statement execution is captured.
- **off** indicates that the CPU time consumed during SQL statement execution is not captured.

**Default value:** on

## query\_log\_file (Deprecated)

**Parameter description:** Specifies the name of a slow query log file on the server. If **enable\_slow\_query\_log** is set to **ON**, slow query records are written into log

files. Only the **sysadmin** user can access this parameter. Generally, log file names are generated in strftime mode. Therefore, the system time can be used to define log file names, which are implemented using the escape character %. This parameter has been deprecated in this version.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

You are advised to use escape character % to specify the log file names for efficient management of log files.

---

**Value range:** a string

**Default value:** `slow_query_log-%Y-%m-%d_%H%M%S.log`

## query\_log\_directory (Deprecated)

**Parameter description:** Specifies the directory for storing low query log files when **enable\_slow\_query\_log** is set to **on**. Only the **sysadmin** user can access this parameter. It can be an absolute path or a relative path (relative to the data directory), which has been deprecated in this version.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

If this parameter is set to an invalid path, the cluster cannot be started.

---

** NOTE**

Valid path: You have read and write permissions on the path.

Invalid path: You do not have read or write permission on the path.

**Value range:** a string

**Default value:** specified during installation

## asp\_log\_directory

**Parameter description:** Specifies the directory for storing ASP log files on the server when **asp\_flush\_mode** is set to **all** or **file**. The value can be an absolute path, or relative to the data directory. Only the **sysadmin** user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

---

**NOTICE**

If this parameter is set to an invalid path, the cluster cannot be started.

---

 **NOTE**

- Valid path: You have read and write permissions on the path.
- Invalid path: You do not have read or write permissions on an invalid path.

**Value range:** a string

**Default value:** specified during installation

## perf\_directory

**Parameter description:** Specifies the directory of the output file of the performance view dotting task. Only the sysadmin user can access this parameter. The value can be an absolute path, or relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

 **NOTE**

- Valid path: You have read and write permissions on the path.
- Invalid path: You do not have read or write permissions on an invalid path.

**Value range:** a string

**Default value:** specified during installation

## enable\_stmt\_track

**Parameter description:** Specifies whether to enable the full/slow SQL statement feature.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU/256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 1.2% by enabling or disabling this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** Full/Slow SQL capture is enabled.
- **off:** Full/Slow SQL capture is disabled.

**Default value:** on

## track\_stmt\_parameter

**Parameter description:** After **track\_stmt\_parameter** is enabled, the executed statements recorded in **statement\_history** are not normalized. The complete SQL statement information can be displayed to help the database administrator locate

faults. For a simple query, the complete statement information is displayed. For a PBE statement, the complete statement information and information about each variable value are displayed. The format is "query string; parameters: \$1=value1,\$2=value2, ...". This parameter is used to display full SQL information for users and is not controlled by the **track\_activity\_query\_size** parameter. When the SQL bypass logic is used for PBE statements, parameters are directly delivered to DNs. Therefore, the number of complete statements cannot be obtained by querying **statement\_history** on CNs. In addition, DNs do not have query character strings. Therefore, complete statement information cannot be obtained by querying **statement\_history** on DNs.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on:** The function of displaying complete SQL statement information is enabled.
- **off:** The function of displaying complete SQL statement information is disabled.

**Default value:** off

## track\_stmt\_session\_slot

**Parameter description:** Specifies the maximum number of full/slow SQL statements that can be cached in a session. If the number of full/slow SQL statements exceeds this value, new statements will not be traced until the flush thread flushes the cached statements to the disk to reserve free space.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 1000

## track\_stmt\_details\_size

**Parameter description:** Specifies the maximum size of execution events that can be collected by a single statement.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 100000000. The unit is byte.

**Default value:** 4096

## track\_stmt\_retention\_time

**Parameter description:** Specifies the retention period of full/slow SQL statement records. This parameter is a combination of parameters. This parameter is read every 60 seconds and records older than the retention period are deleted. Only the **sysadmin** user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string consisting of two parts in the format of 'full sql retention time, slow sql retention time'.

- **full sql retention time** indicates the retention period of full SQL statements. The value ranges from 0 to 86400. The unit is second.
- **slow sql retention time** indicates the retention period of slow SQL statements. The value ranges from 0 to 604800. The unit is second.

**Default value:** 3600,604800

## track\_stmt\_stat\_level

**Parameter description:** Determines the level of statement execution tracing.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#). Letters are case insensitive. If the full SQL function is enabled, the performance will be affected and a large amount of disk space may be occupied.

**Value range:** a string

This parameter consists of two parts in the format of 'full sql stat level, slow sql stat level'.

- The first part indicates the tracing level of full SQL statements. The value can be **OFF**, **L0**, **L1**, or **L2**.
- The second part indicates the tracing level of slow SQL statements. The value can be **OFF**, **L0**, **L1**, or **L2**.

### NOTE

If the tracing level of full SQL statements is not **OFF**, the current SQL statement tracing level is a higher level ( $L2 > L1 > L0$ ) of full and slow SQL statements. For details about the levels, see "System Catalogs and System Views > System Catalogs > STATEMENT\_HISTORY > STATEMENT\_HISTORY columns" in *Developer Guide*.

**Default value:** OFF,L0

## track\_stmt\_standby\_chain\_size

**Parameter description:** Specifies the maximum memory and disk space occupied by fast/slow SQL statement records on the standby node. This parameter is a combination of parameters. This parameter is unavailable in a distributed system.

## 14.3.29 System Performance Snapshot

### enable\_wdr\_snapshot

**Parameter description:** Specifies whether to enable the database monitoring snapshot function.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the database monitoring snapshot function is enabled.

- **off** indicates that the database monitoring snapshot function is disabled.

**Default value:** on

## wdr\_snapshot\_retention\_days

**Parameter description:** Specifies the number of days database monitoring snapshots are retained. If the number of days that the snapshots are retained exceeds the value of this parameter, the system deletes the snapshots with the smallest ID at an interval specified by **wdr\_snapshot\_interval**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 30. The unit is day.

**Default value:** 8

## wdr\_snapshot\_query\_timeout

**Parameter description:** Specifies the execution timeout for the SQL statements associated with database monitoring snapshot operations. If the SQL statement execution is not complete and no result is returned within the specified time, the snapshot operation fails.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#). The value **0** indicates that this parameter does not take effect.

**Value range:** an integer ranging from 0 to 2147483647. The unit is s.

**Default value:** 100s

## wdr\_snapshot\_interval

**Parameter description:** Specifies the interval at which the backend thread Snapshot automatically performs snapshot operations on the database monitoring data.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10 to 60. The unit is min.

**Default value:** 1h

## enable\_asp

**Parameter description:** Specifies whether to enable the active session profile function.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** The function is enabled.
- **off:** The function is disabled.

**Default value:** on

## asp\_sample\_num

**Parameter description:** Specifies the maximum number of samples allowed in the LOCAL\_ACTIVE\_SESSION view. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10000 to 100000

**Default value:** 100000

## asp\_sample\_interval

**Parameter description:** Specifies the sampling interval.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 10. The unit is s.

**Default value:** 1s

## asp\_flush\_rate

**Parameter description:** When the number of samples reaches the value of **asp\_sample\_num**, the samples in the memory are updated to the disk based on a certain proportion. **asp\_flush\_rate** indicates the update proportion. If this parameter is set to **10**, the update ratio is 10:1.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 10

**Default value:** 10

## asp\_flush\_mode

**Parameter description:** Specifies the mode for the ASP to update data to the disk. The value can be **file** (default value), **table** (system catalog), or **all** (system catalog and file). Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, which can be **table**, **file**, or **all**

**Default value:** table

## asp\_retention\_days

**Parameter description:** Specifies the maximum number of days for reserving ASP samples when they are written to the system catalog.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 7. The unit is day.

**Default value:** 2

## asp\_log\_filename

**Parameter description:** Specifies the file name format when writing files using ASP. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** asp-%Y-%m-%d\_%H%M%S.log

## 14.3.30 Security Configuration

### enable\_security\_policy

**Parameter description:** Specifies whether the unified audit and dynamic data masking policies take effect.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**on:** The security policy is enabled.

**off:** The security policy is disabled.

**Default value:** off

### use\_elastic\_search

**Parameter description:** Specifies whether to send unified audit logs to Elasticsearch. If **enable\_security\_policy** and this parameter are enabled, unified audit logs are sent to Elasticsearch through HTTP or HTTPS (used by default). After this parameter is enabled, ensure that the Elasticsearch service corresponding to **elastic\_search\_ip\_addr** can be properly connected. Otherwise, the process fails to be started.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**on:** Unified audit logs are sent to Elasticsearch.

**off:** Unified audit logs are not sent to Elasticsearch.

**Default value:** off

## elastic\_search\_ip\_addr

**Parameter description:** Specifies the IP address of the Elasticsearch system. If HTTPS is used, the format is **https://ip:port.username**. If HTTP is used, the format is **http://ip:port**. In the preceding command, *ip* indicates the IP address of the Elasticsearch server. *port* indicates the listening port for Elasticsearch HTTP communication, and the value ranges from 9200 to 9299. *username* indicates the username used for registering an Elasticsearch account. The initial user is **elastic**. If HTTPS is used, related certificates need to be configured. For details, see "Unified Auditing" in the *Security Hardening Guide*.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

**Default value:** "

## is\_sysadmin

**Parameter description:** Specifies whether the current user is an initial user.

This parameter is a fixed INTERNAL parameter and cannot be modified.

**Value range:** Boolean

**on** indicates that the user is an initial user.

**off** indicates that the user is not an initial user.

**Default value:** off

## block\_encryption\_mode

**Parameter description:** Specifies the block encryption mode used by the `aes_encrypt` and `aes_decrypt` functions for encryption and decryption.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated values. Valid values are **aes-128-cbc**, **aes-192-cbc**, **aes-256-cbc**, **aes-128-cfb1**, **aes-192-cfb1**, **aes-256-cfb1**, **aes-128-cfb8**, **aes-192-cfb8**, **aes-256-cfb8**, **aes-128-cfb128**, **aes-192-cfb128**, **aes-256-cfb128**, **aes-128-ofb**, **aes-192-ofb**, and **aes-256-ofb**. **aes** indicates the encryption or decryption algorithm. **128**, **192**, and **256** indicate the key length (unit: bit). **cbc**, **cfb1**, **cfb8**, **cfb128**, and **ofb** indicate the block encryption or decryption mode.

**Default value:** aes-128-cbc

## 14.3.31 HyperLogLog

### hll\_default\_log2m

**Parameter description:** Specifies the number of buckets for HLL data. The number of buckets affects the precision of distinct values calculated by HLL. The more buckets there are, the smaller the deviation is. The deviation range is as follows:  $[-1.04/2^{\log_2 m^{*1/2}}, +1.04/2^{\log_2 m^{*1/2}}]$

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10 to 16

**Default value:** 14

## hll\_default\_log2explicit

**Parameter description:** Specifies the default threshold for switching from the explicit mode to the sparse mode.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 12 The value 0 indicates that the explicit mode is skipped. The value 1 to 12 indicates that the mode is switched when the number of distinct values reaches  $2^{\text{hll\_default\_log2explicit}}$ .

**Default value:** 10

## hll\_default\_log2sparse

**Parameter description:** Specifies the default threshold for switching from the sparse mode to the full mode.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 14 The value 0 indicates that the explicit mode is skipped. The value 1 to 14 indicates that the mode is switched when the number of distinct values reaches  $2^{\text{hll\_default\_log2sparse}}$ .

**Default value:** 12

## hll\_duplicate\_check

**Parameter description:** Specifies whether duplicatecheck is enabled by default.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or 1 0: disabled; 1: enabled

**Default value:** 0

## hll\_default\_regwidth (Discarded)

**Parameter description:** Specifies the number of bits in each bucket for HLL data. A larger value indicates more memory occupied by HLL. **hll\_default\_regwidth** and **hll\_default\_log2m** determine the maximum number of distinct values that can be calculated by HLL. Currently, **regwidth** is set to a fixed value and is no longer used.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 5

**Default value:** 5

### hll\_default\_expthresh (Discarded)

**Parameter description:** Specifies the default threshold for switching from the **explicit** mode to the **sparse** mode. Currently, the **hll\_default\_log2explicit** parameter is used to replace the similar function.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 7 -1 indicates the auto mode; 0 indicates that the **explicit** mode is skipped; a value from 1 to 7 indicates that the mode is switched when the number of distinct values reaches  $2^{\text{hll\_default\_expthresh}}$ .

**Default value:** -1

### hll\_default\_sparseon (Discarded)

**Parameter description:** Specifies whether to enable the **sparse** mode by default. Currently, the **hll\_default\_log2sparse** parameter is used to replace the similar function. When **hll\_default\_log2sparse** is set to 0, the **sparse** mode is disabled.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or 1 0 indicates that the **sparse** mode is disabled by default. 1 indicates that the **sparse** mode is enabled by default.

**Default value:** 1

### hll\_max\_sparse (Discarded)

**Parameter description:** Specifies the size of **max\_sparse**. Currently, the **hll\_default\_log2sparse** parameter is used to replace the similar function.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to 2147483647

**Default value:** -1

### enable\_compress\_hll (Discarded)

**Parameter description:** Specifies whether to enable memory optimization for HLL. Currently, the HLL memory has been optimized, and this parameter is no longer used.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** or **true** indicates that memory optimization is enabled.
- **off** or **false** indicates that memory optimization is disabled.

**Default value:** off

## 14.3.32 User-defined Functions

### udf\_memory\_limit

**Parameter description:** Specifies the maximum physical memory that can be used when each CN or DN executes UDFs. This parameter does not take effect in the current version. Use **FencedUDFMemoryLimit** and **UDFWorkerMemHardLimit** to control virtual memory used by fenced udf worker.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 200 x 1024 to 2147483647. The unit is KB.

**Default value:** 200 MB

### FencedUDFMemoryLimit

**Parameter description:** Specifies the virtual memory used by each fenced udf worker process.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Setting Suggestions:** For details, see "User-defined Functions > PL/Java Functions" in *Developer Guide*.

**Value range:** an integer ranging from 0 KB to 2147483647 KB. The unit can also be MB or GB. **0** indicates that the memory is not limited.

**Default value:** 0

### UDFWorkerMemHardLimit

**Parameter description:** Specifies the maximum value of **fencedUDFMemoryLimit**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 KB to 2147483647 KB. The unit can also be MB or GB.

**Default value:** 1GB

## 14.3.33 Collaborative Analysis

Due to specification changes, the current version no longer supports this feature. Do not use it.

### enable\_agg\_pushdown\_for\_ca

**Parameter description:** In collaborative analysis, this parameter specifies whether to convert the Agg operator above the ForeignScan operator into a remote SQL statement and send it to a remote cluster.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the Agg operator above ForeignScan is converted into a remote SQL statement.
- **off** indicates that only the ForeignScan operator is converted to a remote SQL statement.

**Default value:** on

### 14.3.34 Acceleration Cluster

Due to specification changes, the current version no longer supports this feature. Do not use it.

#### `show_acce_estimate_detail`

**Parameter description:** When the cluster is accelerated (`acceleration_with_compute_pool` is set to **on**), this parameter specifies whether the **EXPLAIN** statement displays the evaluation information about execution plan pushed down to the accelerated cluster. (Due to specification changes, the current version no longer supports this feature. Do not use it.) The evaluation information is generally used by O&M personnel during maintenance, and it may affect the output display of the **EXPLAIN** statement. Therefore, this parameter is disabled by default. The evaluation information is displayed only if the **verbose** option of the **EXPLAIN** statement is enabled.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the evaluation information is displayed in the output of the **EXPLAIN** statement.
- **off** indicates that the evaluation information is not displayed in the output of the **EXPLAIN** statement.

**Default value:** off

#### `acceleration_with_compute_pool`

**Parameter description:** Specifies whether to use the computing resource pool for acceleration when an OBS is queried. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the query covering the OBS is accelerated based on the cost when the computing resource pool is available.
- **off** indicates that no query is accelerated using the computing resource pool.

**Default value:** off

## max\_resource\_package

**Parameter description:** Specifies the upper limit of the concurrent task threads for accelerating each cluster each DN.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 to 2147483647

**Default value:** 0

## 14.3.35 Scheduled Task

### job\_queue\_processes

**Parameter description:** Specifies the number of jobs that can be concurrently executed. This parameter is a POSTMASTER parameter. You can set it using **gs\_guc**, and you need to restart **gaussdb** to make the setting take effect.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 to 1000

Function:

- Setting **job\_queue\_processes** to **0** indicates that the scheduled job function is disabled and that no job will be executed. (Enabling scheduled jobs may affect the system performance. At sites where this function is not required, you are advised to disable it.)
- Setting **job\_queue\_processes** to a value that is greater than **0** indicates that the scheduled job function is enabled and this value is the maximum number of jobs that can be concurrently processed.

After the scheduled job function is enabled, the **job\_scheduler** thread polls the **pg\_job** system catalog at a scheduled interval. The scheduled job check is performed every second by default.

Too many concurrent jobs consume many system resources, so you need to set the number of concurrent jobs to be processed. If the current number of concurrent jobs reaches the value of **job\_queue\_processes** and some of them expire, these jobs will be postponed to the next polling period. Therefore, you are advised to set the polling interval (the **Interval** parameter of the **submit** API) based on the execution duration of each job to avoid the problem that jobs in the next polling period cannot be properly processed because of overlong job execution time.

Note: If the number of concurrent jobs is large and the value is too small, these jobs will wait in queues. However, a large parameter value leads to large resource consumption. You are advised to set this parameter to **100** and change it based on the system resource condition.

**Default value:** 10

### enable\_prevent\_job\_task\_startup

**Parameter description:** Specifies whether to start the job thread.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the job thread is not started.
- **off** indicates that the job thread is started.

**Default value:** off

### 14.3.36 Thread Pool

The overall design idea of the thread pool technology is to pool thread resources and reuse them among different connections. After the system is started, a fixed number of worker threads are started based on the current number of cores or user configuration. A worker thread serves one or more connection sessions. In this way, the session and thread are decoupled.

#### enable\_thread\_pool

**Parameter description:** Specifies whether to enable the thread pool function. This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the thread pool function is enabled.
- **off** indicates that the thread pool function is disabled.

**Default value:** on

#### thread\_pool\_attr

**Parameter description:** Specifies the detailed attributes of the thread pool function. This parameter is valid only when **enable\_thread\_pool** is set to **on**. Only user **sysadmin** can access this parameter. This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, consisting of one or more characters.

This parameter consists of three parts: **thread\_num**, **group\_num**, and **cpubind\_info**. The meanings of the three parts are as follows:

- **thread\_num** indicates the total number of threads in the thread pool. The value ranges from 0 to 4096. The value **0** indicates that the database automatically configures the number of threads in the thread pool based on the number of CPU cores. If the value is greater than **0**, the number of threads in the thread pool is the same as the value of **thread\_num**. You are advised to set the thread pool size based on the hardware configuration. The formula is as follows: Value of **thread\_num** = *Number of CPU cores* × **3-5**. The maximum value of **thread\_num** is **4096**.
- **group\_num** indicates the number of thread groups in the thread pool. The value ranges from 0 to 64. The value **0** indicates that the database automatically configures the number of thread groups in the thread pool based on the number of NUMA groups. If the value is greater than **0**, the

number of thread groups in the thread pool is the same as the value of **group\_num**.

- **cpubind\_info** indicates whether the thread pool is bound to a core. The available configuration modes are as follows: 1. '(nobind)': The thread is not bound to a core. 2. '(allbind)': Use all CPU cores that can be queried in the current system to bind threads. 3. '(nodebind: 1, 2)': Use the CPU cores in NUMA groups 1 and 2 to bind threads. 4. '(cpubind: 0-30)': Use CPU cores 0 to 30 to bind threads. 5. '(numabind: 0-30)': Use CPU cores 0 to 30 in the NUMA group to bind threads. This parameter is case-insensitive.

**Default value:**

- Independent deployment: '**1024,2,(nobind)**' (60-core CPU/480 GB memory and 32-core CPU/256 GB memory); '**512,2,(nobind)**' (16-core CPU/128 GB memory); '**256,2,(nobind)**' (8-core CPU/64 GB memory); '**128,2,(nobind)**' (4-core CPU/32 GB memory); '**64,2,(nobind)**' (4-core CPU/16 GB memory)

## thread\_pool\_stream\_attr

**Parameter description:** Specifies the detailed attributes of the stream thread pool function. This parameter is valid only when **enable\_thread\_pool** is set to **on** and only takes effect on DNs. Only user **sysadmin** can access this parameter. This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, consisting of one or more characters.

This parameter consists of four parts: **stream\_thread\_num**, **stream\_proc\_ratio**, **group\_num**, **cpubind\_info**. The meanings of the four parts are as follows:

- **stream\_thread\_num** indicates the total number of threads in the stream thread pool. The value ranges from 0 to 4096. The value **0** indicates that the database automatically configures the number of threads in the thread pool based on the number of CPU cores. If the value is greater than **0**, the number of threads in the thread pool is the same as the value of **stream\_thread\_num**. You are advised to set the thread pool size based on the hardware configuration. The formula is as follows: Value of **stream\_thread\_num** = *Number of CPU cores* x **3-5**. The maximum value of **stream\_thread\_num** is **4096**.
- **stream\_proc\_ratio** indicates the ratio of proc resources reserved for stream threads. The value is a floating-point number. The default value is **0.2**. The reserved proc resources are calculated as follows: The value of **stream\_proc\_ratio** x the value of **stream\_thread\_num**.
- **group\_num** indicates the number of thread groups in the thread pool. The value ranges from 0 to 64. The value **0** indicates that the database automatically configures the number of thread groups in the thread pool based on the number of NUMA groups. If the value is greater than **0**, the number of thread groups in the thread pool is the same as the value of **group\_num**. The value of **group\_num** in **thread\_pool\_stream\_attr** must be the same as that in **thread\_pool\_attr**. If they are set to different values, the value of **group\_num** in **thread\_pool\_attr** is used.
- **cpubind\_info** indicates whether the thread pool is bound to a core. The available configuration modes are as follows: 1. '(nobind)': The thread is not bound to a core. 2. '(allbind)': Use all CPU cores that can be queried in the

current system to bind threads. 3. '**(nodebind: 1, 2)**': Use the CPU cores in NUMA groups 1 and 2 to bind threads. 4. '**(cpubind: 0-30)**': Use CPU cores 0 to 30 to bind threads. 5. '**(numabind: 0-30)**': Use CPU cores 0 to 30 in the NUMA group to bind threads. This parameter is case-insensitive. The value of **cpubind\_info** in **thread\_pool\_stream\_attr** must be the same as that in **thread\_pool\_attr**. If they are set to different values, the value of **cpubind\_info** in **thread\_pool\_attr** is used.

**Default value:**

**stream\_thread\_num:** 16

**stream\_proc\_ratio:** 0.2

**group\_num** and **cpubind\_info**: For details, see [thread\\_pool\\_attr](#).

## resilience\_threadpool\_reject\_cond

**Parameter description:** Specifies the percentage of thread pool usage for escape from overload. This parameter takes effect only when the GUC parameters **enable\_thread\_pool** and **use\_workload\_manager** are enabled. This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string, consisting of one or more characters.

This parameter consists of **recover\_threadpool\_percent** and **overload\_threadpool\_percent**. The meanings of the two parts are as follows:

- **recover\_threadpool\_percent:** specifies the thread pool usage when the thread pool recovers to the normal state. When the thread pool usage is less than the value of this parameter, the escape from overload function is disabled and new connections are allowed. The value ranges from 0 to *INT\_MAX*. The value indicates a percentage.
- **overload\_threadpool\_percent:** specifies thread pool usage when the thread pool is overloaded. If the thread pool usage is greater than the value of this parameter, the current thread pool is overloaded. In this case, the escape from overload function is enabled to kill sessions and forbid new connections. The value ranges from 0 to *INT\_MAX*. The value indicates a percentage.

**Default value:** '0,0', indicating that the thread pool escape function is disabled.

**Example:**

```
resilience_threadpool_reject_cond = '50,90'
```

When the thread pool usage exceeds 90%, new connections are forbidden and stacked sessions are killed. When the thread pool usage decreases to 50%, session killing is stopped and new connections are allowed.

#### NOTICE

- The thread pool usage can be queried in the DBE\_PERF.local\_threadpool\_status view. The initial number of threads in the thread pool can be obtained by querying the **thread\_pool\_attr** parameter.
- If this parameter is set to a small value, the thread pool escape from overload process is frequently triggered. As a result, ongoing sessions are forcibly logged out, and new connections fail to be connected for a short period of time. Therefore, exercise caution when setting this parameter based on the actual thread pool usage.
- If the **use\_workload\_manager** parameter is disabled and the **bypass\_workload\_manager** parameter is enabled, this parameter also takes effect. The **bypass\_workload\_manager** parameter is of the SIGHUP type; therefore, after the reload mode is set, you need to restart the database for the setting to take effect.
- The values of **recover\_threadpool\_percent** and **overload\_threadpool\_percent** can be 0 at the same time. In addition, the value of **recover\_threadpool\_percent** must be smaller than that of **overload\_threadpool\_percent**. Otherwise, the setting does not take effect.

## 14.3.37 Full Text Search

### ngram\_gram\_size

**Parameter description:** Specifies the length of the ngram parser segmentation.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 4

**Default value:** 2

### ngram\_grapsymbol\_ignore

**Parameter description:** Specifies whether the ngram parser ignores graphical characters.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** The ngram parser ignores graphical characters.
- **off:** The ngram parser does not ignore graphical characters.

**Default value:** off

### ngram\_punctuation\_ignore

**Parameter description:** Specifies whether the ngram parser ignores punctuation marks.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** The ngram parser ignores punctuation marks.
- **off:** The ngram parser does not ignore punctuation marks.

**Default value:** on

## 14.3.38 Backup and Restoration

### operation\_mode

**Parameter description:** Specifies whether the system enters the backup and restoration mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the system is in the backup and restoration mode.
- **off** indicates that the system is not in the backup and restoration mode.

**Default value:** off

### enable\_cbm\_tracking

**Parameter description:** Specifies whether to enable cbm tracking. To perform full or incremental backup for the cluster by using Roach, set this parameter to **on**. Otherwise, the backup will fail.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on:** The cbm tracking is enabled.
- **off:** The cbm tracking is disabled.

**Default value:** off

### max\_size\_for\_xlog\_retention

**Parameter description:** Specifies when to forcibly push the backup replication slot to prevent the disk from being full and the cluster from being read-only because logs cannot be recycled during backup. It is recommended that the value of this parameter be a little smaller than the value of **datastorage\_threshold\_value\_check** of the CM Server component to prevent the cluster from entering the read-only state.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** -100 to 2147483647

- The value **0** indicates that this function is disabled.
- A negative value indicates that the backup replication slot is forcibly pushed when the disk usage exceeds 80% and log recycling is blocked due to backup operations. For example, **-80** indicates that the backup replication slot is forcibly pushed when the disk usage exceeds 80%.
- A positive value indicates that the backup replication slot is triggered based on the size of stacked logs. For example, **32** indicates that the backup replication slot is forcibly pushed when the redo position of the current checkpoint exceeds 32 log segments (the size of each log segment is 16 MB) and logs are recycled because the backup operation is blocked.

**Default value:** 0

### max\_cbm\_retention\_time

**Parameter description:** Specifies the interval at which CBM backup files are forcibly recycled. If CBM files cannot be recycled during backup, the disk will be full and the cluster will be read-only. You are advised to set this parameter based on the full backup interval.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 86400 to 2147483647

- The unit is second.
- The minimum value is 1 day.
- The default value is 2 weeks.

**Default value:** 1209600

## 14.3.39 AI Features

### enable\_hypo\_index

**Parameter description:** Specifies whether the database optimizer considers the created virtual index when executing the **EXPLAIN** statement. By executing **EXPLAIN** on a specific query statement, you can evaluate whether the index can improve the execution efficiency of the query statement based on the execution plan provided by the optimizer.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on** indicates that a virtual index is created during **EXPLAIN** execution.
- **off** indicates that no virtual index is created during **EXPLAIN** execution.

**Default value:** off

### enable\_ai\_stats

**Parameter description:** Specifies whether to create or use intelligent statistics.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

**Default value:** off

## multi\_stats\_type

**Parameter description:** Specifies the type of statistics to be created when **enable\_ai\_stats** is set to **on**.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** BAYESNET, MCV, or ALL.

- **BAYESNET:** Only intelligent statistics are created.
- **MCV:** Only traditional statistics are created.
- **ALL:** Both traditional statistics and intelligent statistics are created.

**Default value:** BAYESNET

## ai\_stats\_cache\_limit

**Parameter description:** Specifies the maximum number of models that can be cached when **enable\_ai\_stats** is set to **on**.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 30 to 1000

**Default value:** 100

## enable\_operator\_prefer

**Parameter description:** Specifies whether to enable the operator preference rule. If the estimated costs are similar, the parameterized path is preferred for table join. Note: There are two prerequisites for this parameter to take effect: 1. The parameterized path is generated. 2. The estimated cost of the parameterized path is similar to that of other index scan operators.

This is a USERSET parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

**Default value:** off

## enable\_ai\_watchdog

**Parameter description:** Enables or disables the AI watchdog function.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** on

## enable\_ai\_watchdog\_forcible\_oom\_detection

**Parameter description:** Forcibly enables or disables OOM detection of the AI watchdog. If this parameter is disabled, the system automatically determines whether to enable OOM detection based on the current database specifications. In automatic determination mode, OOM detection is enabled only when **max\_process\_memory** is set to 64 GB or larger.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** off

## enable\_ai\_watchdog\_healing

**Parameter description:** Enables or disables the self-healing function of the AI watchdog. You are advised not to enable this function because it is easy to cause suspension in high-load scenarios with 16-core CPU or lower specifications.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** Boolean

- **on:** enabled.
- **off:** disabled.

**Default value:** on

## ai\_watchdog\_max\_cpu\_usage

**Parameter description:** Specifies the expected upper limit of the database CPU usage. The value is normalized based on the multi-core situation. If this parameter is set to **0**, the system does not check the CPU usage.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a floating-point number ranging from 0 to 1.

**Default value:** 0.8

## ai\_watchdog\_oom\_dynamic\_used\_threshold

**Parameter description:** Specifies the expected upper limit of the dynamic memory usage of the database.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a floating-point number ranging from 0 to 1.

**Default value:** 0.95

## ai\_watchdog\_oom\_growth\_confidence

**Parameter description:** Specifies the confidence level of the OOM detection algorithm.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a floating-point number ranging from 0.1 to 1.

**Default value:** 0.95

## ai\_watchdog\_oom\_malloc\_failures

**Parameter description:** Specifies the maximum number of consecutive memory allocation failures tolerated. If the number of consecutive memory allocation failures exceeds this value, the OOM detection function may be triggered.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 32000.

**Default value:** 50

## ai\_watchdog\_oom\_other\_used\_memory\_threshold

**Parameter description:** Specifies the expected upper limit of memory usage of other parts of the database, in MB.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 1 to 32000.

**Default value:** 20480

## ai\_watchdog\_oom\_process\_threshold

**Parameter description:** Specifies the expected percentage of the database process usage to the value of **max\_process\_memory**. When the threshold is reached, memory leakage determination is triggered. The value can be greater than 1.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a floating-point number ranging from 0 to 10.

**Default value:** 1.1

## ai\_watchdog\_oom\_shared\_threshold

**Parameter description:** Specifies the expected upper limit of the shared memory usage of the database.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** a floating-point number ranging from 0 to 1.

**Default value:** 0.4

## ai\_watchdog\_rto\_restriction\_time

**Parameter description:** Specifies the RTO threshold of the AI watchdog self-healing function. If the RTO threshold is exceeded, self-healing is not performed. The unit is second.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 36000.

**Default value:** 600

## ai\_watchdog\_tolerance\_times

**Parameter description:** Specifies the maximum number of consecutive abnormal events that can be tolerated by the AI watchdog before self-healing is started. This parameter can be used to avoid incorrect operations.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 100.

**Default value:** 4

## ai\_watchdog\_tps\_threshold

**Parameter description:** Specifies the lower limit of the expected TPS usage of the database instance. If the TPS usage is lower than the value of this parameter, the exception determination logic is triggered.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 32000.

**Default value:** 2

## ai\_watchdog\_wait\_time

**Parameter description:** Adjusts the waiting time, in seconds. To prevent the database from frequently performing self-healing operations, the database waits for a period of time after startup.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 36000.

**Default value:** 1800

## ai\_watchdog\_warning\_retention

**Parameter description:** Specifies the maximum number of alarm records that the AI watchdog can retain in the `db_perf.ai_watchdog_detection_warnings` view.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 14-2](#).

**Value range:** an integer ranging from 0 to 32000.

**Default value:** 20

## 14.3.40 Global SysCache Parameters

### enable\_global\_syscache

**Parameter description:** Specifies whether to enable the global system cache function. This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the global system cache function is enabled.
- **off** indicates that the global system cache function is disabled.

**Default value:** on

You are advised to use this parameter together with the thread pool parameter. After this parameter is enabled, you are advised to set **wal\_level** of the standby node to **hot\_standby** or higher if you need to access the standby node.

### global\_syscache\_threshold

**Parameter description:** Specifies the maximum memory usage of the global system cache.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

The **enable\_global\_syscache** parameter must be enabled.

**Value range:** an integer ranging from 16384 to 1073741824. The unit is KB.

**Default value:** 163840

Recommended calculation formula: The smaller value of the number of hot databases and the number of threads x Memory size allocated to each database, that is, **global\_syscache\_threshold = min(count(hot dbs),count(threads)) x memofdb**.

The number of hot databases refers to the number of frequently accessed databases. In thread pool mode, the number of threads is the sum of the number of threads in the thread pool and the number of background threads. In non-thread pool mode, the number of hot databases is used.

**memofdb** indicates the average memory allocated to each database. The background noise memory of each database is 2 MB. Each time a table or index is added, 11 KB memory is added.

If this parameter is set to a small value, memory is frequently evicted, and a large number of memory fragments cannot be recycled. As a result, memory control fails.

## 14.3.41 Reserved Parameters

### NOTE

The following parameters are reserved and do not take effect in this version.

acce\_min\_datasize\_per\_thread

cstore\_insert\_mode  
dfs\_partition\_directory\_length  
enable\_fstream  
enable\_hdfs\_predicate\_pushdown  
enable\_orc\_cache  
schedule\_splits\_threshold  
enable\_constraint\_optimization  
enable\_hadoop\_env  
enable\_hypo\_index  
undo\_zone\_count

### Discarded Parameters

max\_query\_retry\_times  
enable\_slow\_query\_log

## 14.3.42 Read Parameters of the Standby Node in a Distributed System

### enable\_standby\_read

**Parameter description:** Specifies whether to enable the read function of the standby node for a session. This parameter is a USERSET parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the read function of the standby node in a distributed system is enabled for the session.
- **off** indicates that the read function of the standby node in a distributed system is disabled for the session.

**Default value:** off

#### NOTE

- This function can be enabled only in GTM-free mode, non-transaction block, and hot-standby mode.
- This parameter can be set only at the session level and cannot be set using `gs_guc`.
- If this parameter is set at a non-session level (for example, manually enabling this parameter in the configuration file), the cluster cannot be started when a parameter conflict occurs. If no parameter conflict occurs, the read function of the standby node in a distributed system is also enabled for background threads such as `autovacuum` and `WorkloadMonitor`. As a result, DDL and DML operations are affected and errors are reported.

## standby\_read\_delay

**Parameter description:** Specifies the maximum difference between the primary and standby nodes when data is read from the standby node. If the difference exceeds the value of this parameter, data cannot be read from the standby node. This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to *INT\_MAX*. The unit is ms. -1 indicates that the difference between the primary and standby nodes does not need to be verified when data is read from the standby node. 0 indicates that data is read from the standby node only when there is no difference between the primary and standby nodes.

**Default value:** 10000000

## standby\_read\_rto

**Parameter description:** Specifies the maximum RTO of the system when data is read from the standby node. If the RTO exceeds the value of this parameter, data cannot be read from the standby node. This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from -1 to *INT\_MAX*. The unit is ms. The value -1 indicates that the system RTO verification is not required when data is read from the standby node. The value 0 indicates that data is read from the standby node only when the RTO of at least one standby node is 0.

**Default value:** 60000

## 14.3.43 Restoring Data on the Standby Node

### standby\_page\_repair

**Parameter description:** Specifies whether to enable page repair during playback on the standby node. This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

- **on** indicates that the standby node automatically detects and repairs pages during playback.
- **off** indicates that the standby node does not automatically detect and repair pages during playback.

**Default value:** off

## 14.3.44 Undo

### undo\_space\_limit\_size

**Parameter description:** Specifies the threshold for forcibly recycling undo space. When the undo space usage reaches 80% of the threshold, forcible recycling starts. You are advised to set **undo\_space\_limit\_size** to a value greater than or equal to that of **undo\_limit\_size\_per\_transaction**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 102400 to 2147483647. The unit is 8 KB.

**Default value:** 256 GB

## undo\_limit\_size\_per\_transaction

**Parameter description:** Specifies the undo space threshold of a single transaction. If the threshold is reached, the transaction is rolled back due to an error. You are advised to set **undo\_limit\_size\_per\_transaction** to a value less than or equal to that of **undo\_space\_limit\_size**. If the value of **undo\_limit\_size\_per\_transaction** is greater than that of **undo\_space\_limit\_size**, the displayed value is the same as the configured value when you run the **show undo\_limit\_size\_per\_transaction** command to query the parameter value. The only difference is that the smaller value between **undo\_space\_limit\_size** and **undo\_limit\_size\_per\_transaction** is used as the actual undo space threshold of a single transaction. If the **undo\_limit\_size\_per\_transaction** is set to a value greater than 1 TB, the system performance and stability may be affected.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 256 to 2147483647. The unit is 8 KB.

**Default value:** 32 GB

## 14.3.45 Rollback Parameters

### max\_undo\_workers

**Parameter description:** Specifies the number of undo worker threads invoked during asynchronous rollback. The parameter setting takes effect after the system is restarted.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 100

**Default value:** 5

## 14.3.46 DCF Parameters Settings

### enable\_dcf

**Parameter description:** Specifies whether to enable the DCF mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean. The value can be **on** or **off**. **on** indicates that the current log replication mode is DCF, and **off** indicates that the current log replication mode is not DCF.

**Default value:** off

## dcf\_ssl

**Parameter description:** This parameter is no longer used. The DCF reuses the GUC parameter [ssl](#).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean. The value can be **on** or **off**. The value **on** indicates that SSL is used, and the value **off** indicates that SSL is not used.

**Default value:** on

## dcf\_config

**Parameter description:** Specifies the customized configuration information during installation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Default value:** a string, which is specified by users during installation

## dcf\_data\_path

**Parameter description:** Specifies the DCF data path.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Default value:** a string, which is the **dcf\_data** directory under the data directory of the DN

## dcf\_log\_path

**Parameter description:** Specifies the DCF log path.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Default value:** a string, which is the **dcf\_log** directory under the data directory of the DN.

## dcf\_node\_id

**Parameter description:** Specifies the ID of the DN where the DCF is located. This parameter is defined by the user during installation and mode switching.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Default value:** an integer, which is specified by users during installation

## dcf\_max\_workers

**Parameter description:** Specifies the number of DCF callback function threads. If the number of nodes exceeds 7, increase the value of this parameter (for example,

to **40**). Otherwise, the primary node may remain in the promoting state and the log replication between the primary and standby nodes has no progress.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10 to 262143

**Default value:** 20

## dcf\_truncate\_threshold

**Parameter description:** Specifies the threshold for a DN to truncate DCF logs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647

**Default value:** 100000

## dcf\_election\_timeout

**Parameter description:** Specifies the timeout interval for selecting the DCF leader and follower. The election timeout interval depends on the status of the network between DNs. If the timeout interval is short and the network quality is poor, timeout occurs. After the network recovers, the election becomes normal. You are advised to set a proper timeout interval based on the current network status. Restriction on the DCF node clock: The maximum clock difference between DCF nodes is less than half of the election timeout period. In DCF manual election mode, to ensure timely CM arbitration, do not modify this parameter. Instead, use the default election timeout period.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 600, in seconds

**Default value:** 3

## dcf\_enable\_auto\_election\_priority

**Parameter description:** Specifies whether the DCF priority can be automatically adjusted. The value **0** indicates that automatic adjustment is not allowed, and the value **1** indicates that automatic adjustment is allowed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** 0 or 1.

**Default value:** 1

## dcf\_election\_switch\_threshold

**Parameter description:** Specifies the DCF threshold for preventing frequent switchover to primary. It is recommended that this parameter be set based on the maximum fault duration acceptable for user services.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647, in seconds.

**Default value:** 0

## dcf\_run\_mode

**Parameter description:** Specifies the DCF election mode. The value **0** indicates the automatic election mode, the value **1** indicates the manual election mode, and the value **2** indicates that the election mode is disabled. Currently, the election mode can be disabled only in minority restoration scenarios. If the election mode is disabled, the database instance will become unavailable.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

Note: The working mode of a cluster can be switched only when the cluster is running properly. Otherwise, the cluster is still abnormal after the switching. The DCF working mode configured in GUC parameters must be the same as that configured by using `cm_ctl`. That is, both DCF working modes must be set to manual or automatic at the same time.

For example, to set the DCF manual mode, run the following command:

```
cm_ctl set --param --server -k dn_arbitrate_mode=quorum
cm_ctl reload --param --server
gs_guc reload -Z datanode -I all -N all -c "dcf_run_mode=1"
```

To set the DCF automatic mode, run the following command:

```
cm_ctl set --param --server -k dn_arbitrate_mode=paxos
cm_ctl reload --param --server
gs_guc reload -Z datanode -I all -N all -c "dcf_run_mode=0"
```

**Value range:** 0, 1, or 2

**Default value:** 1

## dcf\_log\_level

**Parameter description:** Specifies the DCF log level.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

- Disable the log function: **NONE**, indicating that the log function is disabled and cannot be used for the following log levels:
- Enable the log function: **RUN\_ERR|RUN\_WAR|RUN\_INF|DEBUG\_ERR|DEBUG\_WAR|DEBUG\_INF|TRACE|PROFILE|OPER**

You can select a string from the preceding strings and use vertical bars (|) to combine the strings. The log level cannot be left blank.

**Default value:** **RUN\_ERR|RUN\_WAR|DEBUG\_ERR|OPER|RUN\_INF|PROFILE**

## dcf\_log\_backup\_file\_count

**Parameter description:** Specifies the number of DCF run log backups.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 1000

**Default value:** 100

### dcf\_max\_log\_file\_size

**Parameter description:** Specifies the maximum size of a DCF run log file.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 1000, in MB

**Default value:** 10

### dcf\_socket\_timeout

**Parameter description:** Specifies the timeout interval for the DCF communication module to connect to the socket. This parameter takes effect upon the system restart. In an environment where the network quality is poor, if the timeout interval is set to a small value, a connection may fail to be set up. In this case, you need to increase the value.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10 to 600000, in ms

**Default value:** 5000

### dcf\_connect\_timeout

**Parameter description:** Specifies the timeout interval for the DCF communication module to set up a connection. This parameter takes effect upon the system restart. In an environment where the network quality is poor, if the timeout interval is set to a small value, the connection may fail to be set up. In this case, you need to increase the value.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 10 to 600000, in ms

**Default value:** 60000

### dcf\_mec\_fragment\_size

**Parameter description:** Specifies the fragment size of the DCF communication module. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 32 to 10240, in KB

**Default value:** 64

## dcf\_stg\_pool\_max\_size

**Parameter description:** Specifies the maximum size of the memory pool of the DCF storage module. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 32 to 2147483647, in MB

**Default value:** 2048

## dcf\_stg\_pool\_init\_size

**Parameter description:** Specifies the minimum size of the memory pool of the DCF storage module. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 32 to 2147483647, in MB

**Default value:** 32

## dcf\_mec\_pool\_max\_size

**Parameter description:** Specifies the maximum size of the memory pool of the DCF communication module. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 32 to 2147483647, in MB

**Default value:** 200

## dcf\_flow\_control\_disk\_rawait\_threshold

**Parameter description:** Specifies the disk waiting threshold for DCF flow control.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647, in  $\mu$ s

**Default value:** 100000

## dcf\_flow\_control\_net\_queue\_message\_num\_threshold

**Parameter description:** Specifies the threshold for the number of messages in a network queue for DCF flow control.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647

**Default value:** 1024

## dcf\_flow\_control\_cpu\_threshold

**Parameter description:** Specifies the threshold for DCF CPU flow control.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 2147483647, in percentage (%)

**Default value:** 100

## dcf\_mec\_batch\_size

**Parameter description:** Specifies the number of batch messages for DCF communication. When the value is **0**, the DCF automatically adjusts the value based on the network and the amount of data to be written. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 1024

**Default value:** 0

## dcf\_mem\_pool\_max\_size

**Parameter description:** Specifies the maximum DCF memory. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 32 to 2147483647, in MB

**Default value:** 2048

## dcf\_mem\_pool\_init\_size

**Parameter description:** Specifies the initial size of the DCF memory. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 32 to 2147483647, in MB

**Default value:** 32

## dcf\_compress\_algorithm

**Parameter description:** Specifies the compression algorithm for DCF run log transmission. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer

- 0 indicates no compression.
- 1 indicates the ZSTD compression algorithm.
- 2 indicates the LZ4 compression algorithm.

**Default value:** 0

## dcf\_compress\_level

**Parameter description:** Specifies the compression level for DCF log transmission. This parameter takes effect upon the system restart. Before this parameter takes effect, a valid compression algorithm must be configured, that is, the **dcf\_compress\_algorithm** parameter is set.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 22

If compression is disabled, the configured compression level does not take effect.

**Default value:** 1

## dcf\_mec\_channel\_num

**Parameter description:** Specifies the number of DCF communication channels. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 64

**Default value:** 1

## dcf\_rep\_append\_thread\_num

**Parameter description:** Specifies the number of DCF log replication threads. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 1000

**Default value:** 2

## dcf\_mec\_agent\_thread\_num

**Parameter description:** Specifies the number of DCF communication working threads. This parameter takes effect upon the system restart. It is recommended that the value of **dcf\_mec\_agent\_thread\_num** be greater than or equal to 2 x Number of nodes x Value of **dcf\_mec\_channel\_num**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 1000

**Default value:** 10

## dcf\_mec\_reactor\_thread\_num

**Parameter description:** Specifies the number of reactor threads used by the DCF. This parameter takes effect upon the system restart. It is recommended that the ratio of the value of **dcf\_mec\_reactor\_thread\_num** to the value of **dcf\_mec\_agent\_thread\_num** be 1:40.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 100

**Default value:** 1

## dcf\_log\_file\_permission

**Parameter description:** Specifies the attribute of the DCF run log file. The parameter setting takes effect after the system is restarted. This parameter is configured during installation and cannot be modified. To allow other users in the same group to access logs, ensure that all parent directories can be accessed by other users in the same group. That is, if **dcf\_log\_path\_permission** is set to **750**, **dcf\_log\_file\_permission** can only be set to **600** or **640**. If **dcf\_log\_path\_permission** is set to **700**, **dcf\_log\_file\_permission** must be set to **600**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated type. The value can be **600** or **640**.

**Default value:** 600

## dcf\_log\_path\_permission

**Parameter description:** Specifies the attribute of the DCF run log directory. The parameter setting takes effect after the system is restarted. This parameter is configured during installation and cannot be modified. To allow other users in the same group to access the log path, set this parameter to **750**. Otherwise, set this parameter to **700**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** enumerated type. The value can be **700** or **750**.

**Default value:** 700

## dcf\_majority\_groups

**Parameter description:** Sets the DCF policy-based majority function. For a group that requires this parameter, at least one standby node in the group receives logs. That is, there is a synchronous standby node in the group. If nodes are added to or deleted from the DCF instance or the group value of a node in the instance is changed, you need to modify the configuration accordingly.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string

- To disable the policy-based majority function, enter an empty string "".
- To enable the function, configure valid group values separated by commas (.). The group values must exist in **dcf\_config**. For example, if the group values 1 and 2 are added to the DCF policy-based majority configuration, you can set this parameter to "1,2". If the group value does not exist in **dcf\_config** or other characters are configured, the DCF considers the configured group invalid.

**Default value:** empty

---

 CAUTION

If all nodes in a group are faulty after the parameter is configured, you need to remove the group from the parameter list when performing node build operations (node recovery or node replacement without changing the IP address) on a node. After the node recovers, you can configure the group again.

---

## dcf\_node\_id\_map

**Parameter description:** Specifies the dictionary mapping between standby DN names and DCF node IDs. The parameter setting takes effect after the system is restarted. This parameter is configured during installation and cannot be modified. This parameter is used in DCF cluster installation, upgrade, and node replacement scenarios. The value of **standby\_name** configured in the GUC parameter **synchronous\_standby\_names** must be included in this dictionary.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** a string. The configuration format is 'standby\_name1:dcf\_node\_id1,standby\_name2:dcf\_node\_id2'. The values of standby DN names and the corresponding DCF node IDs are separated by commas (.).

**Default value:** empty

## 14.3.47 Flashback

This section describes parameters related to the flashback function. In this version, only the Ustore engine supports flashback, while the Astore engine does not support flashback.

## enable\_recyclebin

**Parameter description:** Specifies whether the recycle bin is enabled or disabled in real time.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** Boolean

**Default value:** off

## recyclebin\_retention\_time

**Parameter description:** Specifies the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 1 to 2147483647. The unit is s.

**Default value:** 15 min (900s)

## undo\_retention\_time

**Parameter description:** Specifies the period for retaining undo logs of earlier versions.

This parameter is a SIGHUP parameter. Set it based on instructions provided in [Table 14-1](#).

**Value range:** an integer ranging from 0 to 259200. The unit is second.

**Default value:** 0

---

 **CAUTION**

1. If this parameter is set to **0** during the Ustore flashback query, the snapshot information at the flashback point will be cleared. In earlier versions, no flashback query can be performed. When a flashback query is performed, the error message "cannot find the restore point" is displayed.
  2. If the time within which the undo logs of earlier versions need to be retained is time1 and the SQL statement execution time for the flashback query is time2, you need to set **undo\_retention\_time** to a value greater than time1 + time2. That is, set **undo\_retention\_time** to a value greater than time1 + time2 + 3s. You are advised to set **undo\_retention\_time** to a value equal to time1 + 1.5 x time2. For example, if you want to retain the logs of earlier versions within the latest 3 hours, and the SQL statement execution time for the flashback query is 1 hour, set **undo\_retention\_time** to a value equal to 3 hours + 1.5 x 1 hour, that is, 4.5 hours.
-